

Carderock Division
Naval Surface Warfare Center
Bethesda, Maryland 20817-5700

NSWCCD-26-TR-1998/xx October 1998

Total Ship Systems Directorate
Research and Development Report

Leading Edge Advanced Prototyping For Ships (LEAPS):

LEAPS/API Reference Manual Version 2.0

by
Richard T. Van Eseltine and Robert Ames

Distribution authorized to the Department of Defense and DoD
contractors only; critical technology; October 1998. Other
requests for this document shall be referred to the Carderock
Division, Naval Surface Warfare Center (Code 20)

TABLE OF CONTENTS

TABLE OF CONTENTS	1
PREFACE	6
GLOBALS	6
BASIC LEAPS TYPE DEFINITIONS.....	6
<i>Char8.....</i>	<i>6</i>
<i>CharString.....</i>	<i>6</i>
<i>CharStringList.....</i>	<i>6</i>
<i>Int32.....</i>	<i>6</i>
<i>Int32List.....</i>	<i>6</i>
<i>KeyList.....</i>	<i>6</i>
<i>Logical.....</i>	<i>6</i>
<i>NameVersionPairList.....</i>	<i>6</i>
<i>Real32.....</i>	<i>6</i>
<i>Real32List.....</i>	<i>7</i>
<i>Real64.....</i>	<i>7</i>
<i>Real64List.....</i>	<i>7</i>
<i>UInt32.....</i>	<i>7</i>
<i>UInt32List.....</i>	<i>7</i>
<i>UniqueldList.....</i>	<i>7</i>
BASIC LEAPS ENUMERATIONS	7
<i>ConnectionItemTypeEnum.....</i>	<i>7</i>
<i>ConnectionTypeEnum.....</i>	<i>7</i>
<i>DiagramTypeEnum.....</i>	<i>7</i>
<i>OrientationEnum.....</i>	<i>8</i>
<i>PropertyDataTypeEnum.....</i>	<i>8</i>
<i>TopologicalViewTypeEnum.....</i>	<i>8</i>
DTNURBS ARRAYS.....	8
<i>DtnurbsArrays.....</i>	<i>8</i>
LEAPS PRIMARY CLASSES.....	9
COMPONENT CLASSES	9
<i>Component.....</i>	<i>9</i>
<i>ComponentPtr.....</i>	<i>12</i>
<i>ComponentPtrList.....</i>	<i>12</i>
<i>ComponentPtrMap.....</i>	<i>12</i>
CONCEPT CLASSES.....	12
<i>Concept.....</i>	<i>12</i>
<i>ConceptPtr.....</i>	<i>16</i>
<i>ConceptPtrMap.....</i>	<i>16</i>
<i>ConceptPtrList.....</i>	<i>17</i>
CONNECTION CLASSES.....	17
<i>Connection.....</i>	<i>17</i>
<i>ConnectionPtr.....</i>	<i>21</i>

<i>ConnectionPtrList</i>	21
<i>ConnectionPtrMap</i>	21
DIAGRAM CLASSES	21
<i>Diagram</i>	21
<i>DiagramPtr</i>	24
<i>DiagramPtrList</i>	24
<i>DiagramPtrMap</i>	24
FACTORY CLASSES	24
<i>Factory</i>	24
<i>FactoryPtr</i>	25
<i>FactoryPtrMap</i>	25
SCENARIO CLASSES	25
<i>Scenario</i>	25
<i>ScenarioPtr</i>	27
<i>ScenarioPtrList</i>	27
<i>ScenarioPtrMap</i>	27
STUDY CLASSES	27
<i>Study</i>	27
<i>StudyPtr</i>	30
<i>StudyPtrList</i>	30
<i>StudyPtrMap</i>	31
SYSTEM CLASSES	31
<i>System</i>	31
<i>SystemPtr</i>	35
<i>SystemPtrList</i>	35
<i>SystemPtrMap</i>	35
LEAPS GEOMETRY OBJECT STRUCTURE (GOBS)	35
CoEDGE CLASSES	35
<i>CoEdge</i>	35
<i>CoEdgePtr</i>	37
<i>CoEdgePtrList</i>	37
<i>CoEdgePtrMap</i>	37
COMMONVIEW CLASSES	37
<i>CommonView</i>	37
<i>CommonViewPtr</i>	42
<i>CommonViewPtrList</i>	42
<i>CommonViewPtrMap</i>	42
CoPOINT CLASSES	42
<i>CoPoint</i>	42
<i>CoPointPtr</i>	43
<i>CoPointPtrList</i>	43
<i>CoPointPtrMap</i>	43
EDGE CLASSES	43
<i>Edge</i>	43
<i>EdgePtr</i>	46
<i>EdgePtrList</i>	46
<i>EdgePtrMap</i>	46
EDGELOOP CLASSES	46
<i>EdgeLoop</i>	46

EdgeLoopPtr.....	48
EdgeLoopPtrList	48
EdgeLoopPtrMap.....	48
FACE CLASSES	48
Face.....	48
FacePtr	50
FacePtrList.....	50
FacePtrMap	50
ORIENTEDCLOSEDSHELL CLASSES	51
OrientedClosedShell	51
OrientedClosedShellPtr.....	52
OrientedClosedShellPtrMap.....	52
OrientedClosedShellPtrList	52
PCURVE CLASSES	53
Pcurve	53
PcurvePtr.....	55
PcurvePtrList	55
PcurvePtrMap	55
PPOINT CLASSES	55
Ppoint	55
PpointPtr.....	57
PpointPtrList	57
PpointPtrMap	57
SOLID CLASSES	57
Solid.....	57
SolidPtr	59
SolidPtrList.....	59
SolidPtrMap	59
STRUCTURE CLASSES	59
Structure	59
StructurePtr.....	70
StructurePtrList.....	70
StructurePtrMap.....	70
SURFACE CLASSES	70
Surface	70
SurfacePtr.....	72
SurfacePtrList	73
SurfacePtrMap.....	73
TOPOLOGICALVIEW CLASSES	73
TopologicalView.....	73
TopologicalViewPtr	77
TopologicalViewPtrList.....	77
TopologicalViewPtrMap	77
LEAPS UTILITY CLASSES.....	77
CONNECTIONITEM CLASSES.....	77
ConnectionItem.....	77
ConnectionItemList	78
CURVE CLASSES.....	78
Curve	78

ERROR CLASS	80
<i>Error.....</i>	80
LOCATION CLASSES	80
<i>CartesianLocation.....</i>	80
<i>CoEdgeLocation</i>	81
<i>CurveLocation.....</i>	82
<i>PcurveLocation</i>	82
<i>SurfaceLocation.....</i>	83
MATERIAL CLASSES	84
<i>Material.....</i>	84
<i>MaterialPtr</i>	86
<i>MaterialPtrList.....</i>	86
<i>MaterialPtrMap</i>	87
<i>MaterialGroup.....</i>	87
<i>MaterialGroupPtr.....</i>	90
<i>MaterialGroupPtrList.....</i>	90
<i>MaterialGroupPtrMap.....</i>	90
NAME CLASSES	91
<i>DateTime</i>	91
<i>Name</i>	91
<i>Note.....</i>	92
NAMEVALUEPAIR CLASSES	93
<i>NameValuePair.....</i>	93
<i>NameValuePairList</i>	93
PROPERTY CLASSES	93
<i>IntegerScalar</i>	93
<i>IntegerSTLVector.....</i>	94
<i>Property</i>	94
<i>PropertyPtr.....</i>	96
<i>PropertyPtrList.....</i>	96
<i>PropertyPtrMap.....</i>	97
<i>PropertyData.....</i>	97
<i>PropertyDataPtr</i>	97
<i>PropertyGroup</i>	97
<i>PropertyGroupPtr.....</i>	99
<i>PropertyGroupPtrList</i>	99
<i>PropertyGroupPtrMap.....</i>	99
<i>RealScalar</i>	99
<i>RealSTLVector</i>	100
<i>SplineData</i>	101
<i>String</i>	101
<i>StringSTLVector</i>	102
SPLINE CLASSES	102
<i>Spline.....</i>	102
<i>SplineDomainVariable.....</i>	103
<i>SplineDomainVariableList</i>	104
<i>SplineRangeVariable</i>	104
<i>SplineRangeVariableList.....</i>	104
TOOL CLASSES	105
<i>Tool.....</i>	105

<i>ToolPtr</i>	105
<i>ToolPtrList</i>	105
<i>ToolPtrMap</i>	106

PREFACE

Not Yet Documented

GLOBALS

The Globals Package contains type definitions, enumerations, and constant definitions used through out LEAPS.

Basic LEAPS Type Definitions

Char8

Char8 is a type definition for a character that is represented by 8 bits. It is defined as follows: `char`

CharString

CharString is a type definition for a string. It is defined as follows: `std::string`

CharStringList

CharStringList is a type definition for a list of CharStrings. CharStringList is defined as follows: `std::vector<CharString>`.

Int32

Int32 is a type definition for an integer that is represented by 32 bits. It is defined as follows: `int`

Int32List

Int32List is a type definition for a list of integer numbers. The integers are represented by 32 bits. It is defined as follows: `std::vector<Int32>`.

KeyList

KeyList is a type definition for a list of keys. The keys are represented by a `std::string`. It is defined as follows: `std::vector<std::string>`.

Logical

Logical is a type definition for a boolean. It is defined as follows: `bool`

NameVersionPairList

NameVersionPairList is a type definition for a list of name-version pairs. It is defined as follows: `std::vector<std::pair<std::string, Uint32> >`.

Real32

Real32 is a type definition for a real number that is represented by 32 bits. It is defined as follows: `float`

Real32List

Real32List is a type definition for a list of real numbers. The real numbers are represented by 32 bits. It is defined as follows: `std::vector<Real32>`.

Real64

Real64 is a type definition for a real number that is represented by 64 bits. It is defined as follows: `double`

Real64List

Real64List is a type definition for a list of real numbers. The real numbers are represented by 64 bits. It is defined as follows: `std::vector<Real64>`.

UInt32

UInt32 is a type definition for an unsigned integer that is represented by 32 bits. It is defined as follows: `unsigned int`

UInt32List

UInt32List is a type definition for a list of unsigned integer numbers. The unsigned integers are represented by 32 bits. It is defined as follows: `std::vector<UInt32>`.

UniqueldList

UniqueldList is a type definition for a list of unique identifiers. The unique identifiers are represented by a `std::string`. UniqueldList is defined as follows: `std::vector<std::string>`.

Basic LEAPS Enumerations

ConnectionItemTypeEnum

ConnectionItemTypeEnum is a type definition for an enumeration that specifies the kind of connection item. A connection item is either a Component, System, or Connection object. It is defined as follows:

```
enum { UndefinedItemType, ComponentItemType, SystemItemType,
      ConnectionItemType}
```

ConnectionTypeEnum

ConnectionTypeEnum is a type definition for an enumeration that specifies the kind of connection. It is defined as follows:

```
enum { UnknownConnection = 0, SerialConnection = 10,
      ParallelConnection = 20}
```

DiagramTypeEnum

DiagramTypeEnum is a type definition for an enumeration that specifies the kind of diagram. It is defined as follows:

```
enum { UndefinedDiagram = 0, FunctionalDiagram = 10,
      PhysicalDiagram = 20}
```

OrientationEnum

OrientationEnum is a type definition for an enumeration that specifies the orientation. It is defined as follows:

```
enum { UnknownOrientation = 0, ClockwiseOrientation = -1,  
      CounterClockwiseOrientation = 1, InwardOrientation = -1,  
      OutwardOrientation = 1 }
```

PropertyDataTypeEnum

PropertyDataTypeEnum is a type definition for an enumeration that specifies the type of property data that is stored. It is defined as follows:

```
enum { UndefinedPropertyData=0,  
      RealScalarData=10, RealSTLVectorData=11,  
      IntegerScalarData=20, IntegerSTLVectorData=21,  
      StringData=30, StringSTLVectorData=31, SplineData=40 }
```

TopologicalViewTypeEnum

TopologicalViewTypeEnum is a type definition for an enumeration that specifies the kind of GOBS object (surface, face, or solid) that represents the TopologicalView.

```
enum { UnknownObject = 0, SurfaceObject = 1, FaceObject = 2,  
      SolidObject = 3 }
```

DTNURBS Arrays

The DTNURBS attributes are public and globally accessible. However, the user should **never** modify these attributes because corruption will result. These attributes can be used with function calls to the DTNURBS Spline Library.

DtnurbsArrays

Public Attributes:

char* cmem

Pointer DTNURBS character memory array.

double* dmem

Pointer to DTNURBS double memory array.

int* imem

Pointer DTNURBS integer memory array.

int maxCmem

The size of DTNURBS character memory array.

LEAPS PRIMARY CLASSES

Component Classes

Component

The purpose of the Component class is represent objects that can be thought of as parts of the concept. For example, a computer console or a gun could be considered as "components" of the concept. The component has geometry, properties, and property groups.

Public Attributes:

Name *id* ()

Name object that identifies the Component object.

CartesianLocation *location* ()

CartesianLocation (x,y,z) which specifies the location of the geometric centroid of the Component object in the concept frame of reference.

std::string *name* ()

Name of the Component object.

UInt32 *numberOfConnectionsUsingComponent* ()

Number of Connection objects that use this Component object.

UInt32 *numberOfNotes* ()

Number of Note objects associated with the Component object.

UInt32 *numberOfProperties* ()

Number of Property objects associated with this object.

UInt32 *numberOfPropertyGroups* ()

Number of PropertyGroup objects associated with Component object.

UInt32 *numberOfSystemsUsingComponent* ()

Number of System objects that use the Component object.

Real64* *orientation* ()

Orientation matrix which specifies the orientation of the Component object in the concept frame of reference.

std::string& *uniqueId* ()

Unique identifier of the Component object.

UInt32 *version* ()

Version number of the Component object.

Public Operations:

void *addNote* (const std::string& key, const std::string& comment)

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the Component object.

PropertyPtr *createProperty* (const std::string& name, Uint32 version, const PropertyDataPtr& propData)

Creates a Property object with the given name, version, and property data. This object is associated with the Component object.

PropertyGroupPtr *createPropertyGroup* (const std::string& name, Uint32 version, const PropertyPtrList& propertyData, const PropertyGroupPtrList& propertyGroupData)

Creates a PropertyGroup object with the given name, version, and a list of reference pointers to PropertyGroup and Property objects. This PropertyGroup object is associated with the Component object.

void *debugPrint* ()

Prints the Component object's member values to the error file cerr.

void *destroyComponentStructure* ()

Destroys the current Structure object that represents the geometric views of the Component object and creates a new Structure object to represent the Component object.

void *destroyProperty* (const std::string& name, Uint32 version = 0)

Destroys the Property object with the given name and version that is contained by this Component object. If only the name is given, it is assumed to be the unique identifier of the Property object.

void *destroyPropertyGroup* (const std::string& name, Uint32 version = 0)

Destroys the PropertyGroup object with the given name and version that is contained by this Component object. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

Logical *doesNoteExist* (const std::string& key)

Returns true if the Note object with the given key is associated with the Component object, else false

Logical *doesPropertyExist* (const std::string& name, Uint32 version = 0)

Returns true if the Property object with the given name and version number is associated with the Component object, else false. If only the name is given, it is assumed to be the unique identifier of the Property object.

Logical *doesPropertyGroupExist* (const std::string& name, Uint32 version = 0)

Returns true if the PropertyGroup object with the given name and version number is associated with the Component object, else false. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

StructurePtr *getComponentStructure* ()

Returns a StructurePtr to the Structure object that represents the geometric views of the Component object.

ConnectionPtr *getConnectionUsingComponent* (const std::string& name, Uint32 version = 0)

Returns a ConnectionPtr to the Connection object, given the specified name and version, that uses this Component object. If only the name is given, it is assumed to be the unique id of the Connection object.

const ConnectionPtrList& *getConnectionsUsingComponent* ()

Returns a ConnectionPtrList of all Connection objects that use this Component object.

const KeyList& *getKeysOfNotes* ()

Returns the keys of Note objects associated with the Component object.

void *getNameAndVersion* (std::string& nameOut, Uint32& versionOut)

Returns the name and version number of the Component object in the given arguments.

const NameVersionPairList& *getNameVersionPairsOfProperties* ()

Returns the NameVersionPairs of Property objects that are associated with the Component object.

const NameVersionPairList& *getNameVersionPairsOfPropertyGroups* ()

Returns the NameVersionPairs of PropertyGroup objects that are associated with the Component object.

const Note& *getNote* (const std::string& key)

Returns the Note object with the given key.

PropertyPtr *getProperty* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the Property object with the given name and version number that is contained by this Component object. If only the name is given, it is assumed to be the unique identifier of the Property object.

PropertyGroupPtr *getPropertyGroup* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the PropertyGroup object with the given name and version number that is contained by this Component object. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

SystemPtr *getSystemUsingComponent* (const std::string& name, Uint32 version = 0)

Returns a SystemPtr to the System object, given the specified name and version, that is uses the Component object. If only the name is given, it is assumed to be the unique id of the System object.

const SystemPtrList& *getSystemsUsingComponent* ()

Returns a SystemPtrList of all System objects that use this Component object.

const UniqueIdList& *getUidsOfConnectionsUsingComponent* ()

A list of unique ids of the Connection objects that use this Component object.

const UniqueIdList& *getUidsOfProperties* ()

Returns the unique identifies of Property objects that are associated with the Component object.

const UniqueIdList& *getUidsOfPropertyGroups* ()

Returns the unique identifies of PropertyGroup objects that are associated with the Component object.

const UniqueIdList& *getUidsOfSystemsUsingComponent* ()

A list of unique ids of the System objects that use this Component object.

void *modifyNote* (const std::string& key, const std::string& comment)

Modifies the Note object with the given key, that is associated with the Component object by replacing the comment with the given comment.

void *removeNote* (const std::string& key)

Removes the Note object with the given key from the list of Note objects associated with the Component object.

ComponentPtr

ComponentPtr is a type definition for a reference pointer to a Component object.

ComponentPtrList

ComponentPtrList is a type definition for a list of reference pointers to Component objects.

ComponentPtrMap

ComponentPtrMap is a type definition for an associative map between unique identifiers of Component objects and their reference pointers.

Concept Classes

Concept

The Concept class represents a single concept that is evaluated as part of the study of the IPT. The Concept class is composed of properties, property groups, components, systems, and the structure of the concept.

Public Attributes:

Name *id* ()

Name object that identifies the Concept object.

std::string *name* ()

Name of the Concept object.

UInt32 *numberOfComponents* ()

Number of Component objects that are a part of the Concept object.

UInt32 *numberOfConnections* ()

Number of Connection objects that are associated with the Concept object.

UInt32 *numberOfDiagrams* ()

Number of Diagram objects that are associated with the Concept object.

UInt32 *numberOfNotes* ()

Number of Note objects associated with the Concept object.

UInt32 *numberOfProperties* ()

Number of Property objects associated with the Concept object.

UInt32 *numberOfPropertyGroups* ()

Number of PropertyGroup objects associated with the Concept object.

UInt32 *numberOfSystems* ()

Number of System objects that are part of the Concept object.

std::string& *uniqueId* ()

Unique identifier of the Concept object.

UInt32 *version* ()

Version number of the Concept object.

Public Operations:

void *addNote* (const std::string& key, const std::string& comment)

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the Concept object.

ComponentPtr *createComponent* (const std::string& name, Uint32 version)

Creates a Component object with the given name and version number. This Component object is made a part of the Concept object and a reference pointer (ComponentPtr) is returned.

ConnectionPtr *createConnection* (const std::string& name, Uint32 version)

Creates a Connection object with the given name and version number. This Connection object is made a part of the Concept object and a reference pointer (ConnectionPtr) is returned.

ConnectionPtr *createConnection* (const std::string& name, Uint32 version, const ConnectionItemList& connectionItemList)

Creates a Connection object with the given name, version number, and ConnectionItem list. This Connection object is made a part of the Concept object and a reference pointer (ConnectionPtr) is returned.

DiagramPtr *createDiagram* (const std::string& name, Uint32 version)

Creates a Diagram object with the given name and version number. This Diagram object is made a part of the Concept object and a reference pointer (DiagramPtr) is returned.

DiagramPtr *createDiagram* (const std::string& name, Uint32 version, const ConnectionPtr& connection)

Creates a Diagram object with the given name, version number, and Connection object. This Diagram object is made a part of the Concept object and a reference pointer (DiagramPtr) is returned.

PropertyPtr *createProperty* (const std::string& name, Uint32 version, const PropertyDataPtr& propData)

Creates a Property object with the given name, version, and property data. This object is associated with the Concept object.

PropertyGroupPtr *createPropertyGroup* (const std::string& name, Uint32 version, const PropertyPtrList& propertyData, const PropertyGroupPtrList& propertyGroupData)

Creates a PropertyGroup object with the given name, version, and a list of reference pointers to PropertyGroup and Property objects. This PropertyGroup object is associated with the Concept object.

SystemPtr *createSystem* (const std::string& name, Uint32 version)

Creates a System object with the given name and version number. This System object is made a part of the Concept object and a reference pointer (SystemPtr) is returned.

SystemPtr *createSystem* (const std::string& name, Uint32 version, const ComponentPtrList& componentList, const SystemPtrList& systemList)

Creates a System object with the given name and version number. This System object is made a part of the Concept object and a reference pointer (SystemPtr) is returned.

void *debugPrint* ()

Prints the Concept object's member values to the error file cerr.

void *destroyComponent* (const std::string& name, Uint32 version = 0)

Destroys the Component object with the given name and version that is contained by this Concept object. If only the name is given, it is assumed to be the unique identifier of the Component object.

void *destroyConceptStructure* ()

Destroys the current Structure object that represents the geometric views of the Concept object and creates a new Structure object to represent the Concept object.

void *destroyConnection* (const std::string& name, Uint32 version = 0)

Destroys the Connection object with the given name and version that is contained by this Concept object. If only the name is given, it is assumed to be the unique identifier of the Connection object.

void *destroyDiagram* (const std::string& name, Uint32 version = 0)

Destroys the Diagram object with the given name and version that is contained by this Concept object. If only the name is given, it is assumed to be the unique identifier of the Diagram object.

void *destroyProperty* (const std::string& name, Uint32 version = 0)

Destroys the Property object with the given name and version that is contained by this Concept object. If only the name is given, it is assumed to be the unique identifier of the Property object.

void *destroyPropertyGroup* (const std::string& name, Uint32 version = 0)

Destroys the PropertyGroup object with the given name and version that is contained by this Concept object. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

void *destroySystem* (const std::string& name, Uint32 version = 0)

Destroys the System object with the given name and version that is contained by this Concept object. If only the name is given, it is assumed to be the unique identifier of the System object.

Logical *doesNoteExist* (const std::string& key)

Returns true if the Note object with the given key is associated with the Concept object, else false

Logical *doesComponentExist* (const std::string& name, Uint32 version = 0)

Returns true if the Component object with the given name and version number is a part of the Concept object, else false. If only the name is given, it is assumed to be the unique identifier of the Component object.

Logical *doesConnectionExist* (const std::string& name, Uint32 version = 0)

Returns true if the Connection object with the given name and version number is a part of the Concept object, else false. If only the name is given, it is assumed to be the unique identifier of the Connection object.

Logical *doesDiagramExist* (const std::string& name, Uint32 version = 0)

Returns true if the Diagram object with the given name and version number is a part of the Concept object, else false. If only the name is given, it is assumed to be the unique identifier of the Diagram object.

Logical *doesPropertyExist* (const std::string& name, Uint32 version = 0)

Returns true if the Property object with the given name and version number is associated with the Concept object, else false. If only the name is given, it is assumed to be the unique identifier of the Property object.

Logical *doesPropertyGroupExist* (const std::string& name, Uint32 version = 0)

Returns true if the PropertyGroup object with the given name and version number is associated with the Concept object, else false. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

Logical *doesSystemExist* (const std::string& name, Uint32 version = 0)

Returns true if the System object with the given name and version number is a part of the Concept object, else false. If only the name is given, it is assumed to be the unique identifier of the System object.

ComponentPtr *GetComponent* (const std::string& name, Uint32 version = 0)

Returns a ComponentPtr to the Component object, which is specified by the given name and version, that is part of Concept object. If only the name is given, it is assumed to be the unique id of the Component object.

StructurePtr *getConceptStructure* ()

Returns a StructurePtr to the Structure object that represents the geometric views of the Concept object.

ConnectionPtr *getConnection* (const std::string& name, Uint32 version = 0)

Returns a ConnectionPtr to the Connection object, which is specified by the given name and version, that is part of Concept object. If only the name is given, it is assumed to be the unique id of the Connection object.

DiagramPtr *getDiagram* (const std::string& name, Uint32 version = 0)

Returns a DiagramPtr to the Diagram object, which is specified by the given name and version, that is part of Concept object. If only the name is given, it is assumed to be the unique id of the Diagram object.

const KeyList& *getKeysOfNotes* ()

Returns the keys of Note objects associated with the Concept object.

void *getNameAndVersion* (std::string& nameOut, Uint32& versionOut)

Returns the name and version number of the Concept object in the given arguments.

const NameVersionPairList& *getNameVersionPairsOfComponents* ()

Returns the NameVersionPairs of Component objects that are part of the Concept object.

const NameVersionPairList& *getNameVersionPairsOfConnections* ()

Returns the NameVersionPairs of Connection objects that are part of the Concept object.

const NameVersionPairList& *getNameVersionPairsOfDiagrams* ()

Returns the NameVersionPairs of Diagrams objects that are part of the Concept object.

const NameVersionPairList& *getNameVersionPairsOfProperties* ()

Returns the NameVersionPairs of Property objects that are associated with the Concept object.

const NameVersionPairList& *getNameVersionPairsOfPropertyGroups* ()

Returns the NameVersionPairs of PropertyGroup objects that are associated with the Concept object.

const NameVersionPairList& *getNameVersionPairsOfSystems* ()

Returns the NameVersionPairs of System objects that are part of the Concept object.

const Note& *getNote* (const std::string& key)

Returns the Note object with the given key.

PropertyPtr *getProperty* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the Property object with the given name and version number that is contained by this Concept object. If only the name is given, it is assumed to be the unique identifier of the Property object.

PropertyGroupPtr *getPropertyGroup* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the PropertyGroup object with the given name and version number that is contained by this Concept object. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

SystemPtr *getSystem* (const std::string& name, Uint32 version = 0)

Returns a SystemPtr to the System object, which is specified by the given name and version, that is part of Concept object. If only the name is given, it is assumed to be the unique id of the System object.

const UniqueIdList& *getUidsOfComponents* ()

Returns the unique identifies of Component objects that are part of the Concept object.

const UniqueIdList& *getUidsOfConnections* ()

Returns the unique identifies of Connection objects that are part of the Concept object.

const UniqueIdList& *getUidsOfDiagrams* ()

Returns the unique identifies of Diagram objects that are part of the Concept object.

const UniqueIdList& *getUidsOfProperties* ()

Returns the unique identifies of Property objects that are associated with the Concept object..

const UniqueIdList& *getUidsOfPropertyGroups* ()

Returns the unique identifies of PropertyGroup objects that are associated with the Concept object.

const UniqueIdList& *getUidsOfSystems* ()

Returns the unique identifies of System objects that are part of the Concept object.

void *modifyNote* (const std::string& key, const std::string& comment)

Modifies the Note object with the given key, that is associated with the Concept object, by replacing the comment with the given comment.

void *removeNote* (const std::string& key)

Removes the Note object with the given key from the list of Note objects associated with the Concept object.

ConceptPtr

ConceptPtr is a type definition for a reference pointer to a Concept object.

ConceptPtrMap

ConceptPtrMap is a type definition for an associative map between unique identifiers of Concept objects and their reference pointers. It is defined as follows:

`std::map<std::string, ConceptPtr>`

ConceptPtrList

ConceptPtrList is a type definition for a list of reference pointers to Concept objects.. It is defined as follows:

`std::vector<ConceptPtr>`.

Connection Classes

Connection

The Connection class provides information on how the contained Component, System, and Connection objects are connected (i.e. serial or parallel).

Public Attributes:

ConnectionTypeEnum *connectionType* ()

Returns the connection type (parallel or serial)

void *connectionType* (ConnectionTypeEnum type)

Sets the connection type to be parallel or serial

Name *id* ()

Name object that identifies the Connection object.

std::string *name* ()

Name of the Connection object.

UInt32 *numberOfComponents* ()

Number of Component objects that are part of the Connection object.

UInt32 *numberOfConnectionItems* ()

Number of items (systems, components and/or connections) that are part of the Connection object.

UInt32 *numberOfConnections* ()

Number of Connection objects that are part of the Connection object.

UInt32 *numberOfConnectionsUsingConnection* ()

Number of Connection objects that use this Connection object.

UInt32 *numberOfNotes* ()

Number of Note objects associated with the Connection object.

UInt32 *numberOfProperties* ()

Number of Property objects associated with the Connection object.

UInt32 *numberOfPropertyGroups* ()

Number of PropertyGroup objects associated with the Connection object.

UInt32 *numberOfSystems* ()

Number of System objects that are part of the Connection object.

UInt32 *operationalNumber* ()

Operational number is the number of connection items that are needed for a parallel connection to be operational.

std::string& *uniqueId* ()

Unique identifier of the Connection object.

UInt32 version ()

Version number of the Connection object.

Public Operations:

void addItem (const ComponentPtr& componentToAdd)

Adds the association of the given Component object with the Connection object.

void addItem (const ConnectionPtr& connectionToAdd)

Adds the association of the given Connection object with the Connection object.

void addItem (const SystemPtr& systemToAdd)

Adds the association of the given System object with the Connection object.

void addNote (const std::string& key, const std::string& comment)

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the Connection object.

PropertyPtr createProperty (const std::string& name, UInt32 version, const PropertyDataPtr& propData)

Creates a Property object with the given name, version, and property data. This object is associated with the Connection object.

PropertyGroupPtr createPropertyGroup (const std::string& name, UInt32 version, const PropertyPtrList& propertyData, const PropertyGroupPtrList& propertyGroupData)

Creates a PropertyGroup object with the given name, version, and a list of reference pointers to PropertyGroup and Property objects. This PropertyGroup object is associated with the Connection object.

void debugPrint ()

Prints the Connection object's member values to the error file cerr.

void destroyProperty (const std::string& name, UInt32 version = 0)

Destroys the Property object with the given name and version that is contained by this Connection object. If only the name is given, it is assumed to be the unique identifier of the Property object.

void destroyPropertyGroup (const std::string& name, UInt32 version = 0)

Destroys the PropertyGroup object with the given name and version that is contained by this Connection object. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

Logical doesPropertyExist (const std::string& name, UInt32 version = 0)

Returns true if the Property object with the given name and version number is associated with the Connection object, else false. If only the name is given, it is assumed to be the unique identifier of the Property object.

Logical doesPropertyGroupExist (const std::string& name, UInt32 version = 0)

Returns true if the PropertyGroup object with the given name and version number is associated with the Connection object, else false. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

Logical doesNoteExist (const std::string& key)

Returns true if the Note object with the given key is associated with the Connection object, else false

ComponentPtr *GetComponent* (const std::string& name, Uint32 version = 0)

Returns a ComponentPtr to the Component object, which is specified by the given name and version, that is part of Connection object. If only the name is given, it is assumed to be the unique id of the Component object.

const ComponentPtrList& *getComponents* ()

Returns a ComponentPtrList of all Component objects that compose the Connection object.

ConnectionPtr *getConnection* (const std::string& name, Uint32 version = 0)

Returns a ConnectionPtr to the Connection object, which is specified by the given name and version, that is part of Connection object. If only the name is given, it is assumed to be the unique id of the Connection object.

const ConnectionItemList& *getConnectionItems* ()

Returns a ConnectionItemList of all items (systems, components, and/or connctions) that compose the Connection object.

const ConnectionPtrList& *getConnections* ()

Returns a ConnectionPtrList of all Connection objects that compose the Connection object.

const ConnectionPtrList& *getConnectionsUsingConnection* ()

Returns a ConnectionPtrList of all Connection objects that use this Connection object.

ConnectionPtr *getConnectionUsingConnection* (const std::string& name, Uint32 version = 0)

Returns a ConnectionPtr to the Connection object, given the specified name and version, that uses this Connection object. If only the name is given, it is assumed to be the unique id of the Connection object.

DiagramPtr *getDiagramUsingConnection* ()

Returns a DiagramPtr to the Diagram object, that uses this Connection object as its root connection. If the Connection object is not a root connection, an invalid DiagramPtr is returned.

const KeyList& *getKeysOfNotes* ()

Returns the keys of Note objects associated with the Connection object.

void *getNameAndVersion* (std::string& nameOut, Uint32& versionOut)

Returns the name and version number of the Connection object in the given arguments.

const NameVersionPairList& *getNameVersionPairsOfProperties* ()

Returns the NameVersionPairs of Property objects that are associated with the Connection object.

const NameVersionPairList& *getNameVersionPairsOfPropertyGroups* ()

Returns the NameVersionPairs of PropertyGroup objects that are associated with the Connection object.

const Note& *getNote* (const std::string& key)

Returns the Note object with the given key.

PropertyPtr *getProperty* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the Property object with the given name and version number that is contained by this Connection object. If only the name is given, it is assumed to be the unique identifier of the Property object.

PropertyGroupPtr *getPropertyGroup* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the PropertyGroup object with the given name and version number that is contained by this Connection object. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

SystemPtr *getSystem* (const std::string& name, Uint32 version = 0)

Returns a SystemPtr to the System object, which is specified by the given name and version, that is part of Connection object. If only the name is given, it is the unique id of the System object.

const SystemPtrList& *getSystems* ()

Returns a SystemPtrList of all System objects that compose the Connection object.

const UniqueIdList& *getUidsOfComponents* ()

A list of unique ids of the Component objects that compose the Connection object.

const UniqueIdList& *getUidsOfConnections* ()

A list of unique ids of the Connection objects that compose the Connection object.

const UniqueIdList& *getUidsOfConnectionsUsingConnection* ()

A list of unique ids of the Connection objects that use this Connection object.

const std::string& *getUidOfDiagramUsingConnection* ()

If the Connection object is used as the root connection of a Diagram object, the unique id of the Diagram object is returned. If the Connection object is not used as a root connection an empty string is returned.

const UniqueIdList& *getUidsOfProperties* ()

Returns the unique identifies of Property objects that are associated with the Connection object.

const UniqueIdList& *getUidsOfPropertyGroups* ()

Returns the unique identifies of PropertyGroup objects that are associated with the Connection object.

const UniqueIdList& *getUidsOfSystems* ()

A list of unique ids of the System objects that compose the Connection object.

void *modifyNote* (const std::string& key, const std::string& comment)

Modifies the Note object with the given key, that is associated with the Connection object, by replacing the comment with the given comment.

void *removeItem* (const ComponentPtr& componentToRemove)

Removes the association of the given Component object with the Connection object.

void *removeItem* (const SystemPtr& systemToRemove)

Removes the association of the given System object with the Connection object.

void *removeItem* (const ConnectionPtr& connectionToRemove)

Removes the association of the given Connection object with the Connection object.

void *removeNote* (const std::string& key)

Removes the Note object with the given key from the list of Note objects associated with the Connection object.

ConnectionPtr

ConnectionPtr is a type definition for a reference pointer to a Connection object.

ConnectionPtrList

ConnectionPtrList is a type definition for a list of reference pointers to Connection objects.. It is defined as follows:

```
std::vector<ConnectionPtr>.
```

ConnectionPtrMap

ConnectionPtrMap is a type definition for an associative map between unique identifiers of Connection objects and their reference pointers.. It is defined as follows:

```
std::map<std::string, ConnectionPtr>
```

Diagram Classes

Diagram

The Diagram class provides the schematic information about a system.

Public Attributes:

DiagramTypeEnum **diagramType ()**

Returns the type of the diagram.

void **diagramType (DiagramTypeEnum type)**

Sets the diagram type to be functional or physical.

Name **id ()**

Name object that identifies the Diagram object.

std::string **name ()**

Name of the Diagram object.

UInt32 **numberOfNotes ()**

Number of Note objects associated with the Diagram object.

UInt32 **numberOfProperties ()**

Number of Property objects associated with the Diagram object.

UInt32 **numberOfPropertyGroups ()**

Number of PropertyGroup objects associated with the Diagram object.

UInt32 **numberOfSystemsUsingDiagram ()**

Number of System objects that use the Diagram object.

std::string& **uniqueId ()**

Unique identifier of the Diagram object.

UInt32 **version ()**

Version number of the Diagram object.

Public Operations:

void *addNote* (const std::string& key, const std::string& comment)

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the Diagram object.

void *addRootConnection* (const ConnectionPtr& rootConnection)

Adds the given Connection object as the root connection of this Diagram object.

PropertyPtr *createProperty* (const std::string& name, Uint32 version, const PropertyDataPtr& propData)

Creates a Property object with the given name, version, and property data. This object is associated with the Diagram object.

PropertyGroupPtr *createPropertyGroup* (const std::string& name, Uint32 version, const PropertyPtrList& propertyData, const PropertyGroupPtrList& propertyGroupData)

Creates a PropertyGroup object with the given name, version, and a list of reference pointers to PropertyGroup and Property objects. This PropertyGroup object is associated with the Diagram object.

void *debugPrint* ()

Prints the Diagram object's member values to the error file cerr.

void *destroyProperty* (const std::string& name, Uint32 version = 0)

Destroys the Property object with the given name and version that is contained by this Diagram object. If only the name is given, it is assumed to be the unique identifier of the Property object.

void *destroyPropertyGroup* (const std::string& name, Uint32 version = 0)

Destroys the PropertyGroup object with the given name and version that is contained by this Diagram object. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

Logical *doesNoteExist* (const std::string& key)

Returns true if the Note object with the given key is associated with the Diagram object, else false

Logical *doesPropertyExist* (const std::string& name, Uint32 version = 0)

Returns true if the Property object with the given name and version number is associated with the Diagram object, else false. If only the name is given, it is assumed to be the unique identifier of the Property object.

Logical *doesPropertyGroupExist* (const std::string& name, Uint32 version = 0)

Returns true if the PropertyGroup object with the given name and version number is associated with the Diagram object, else false. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

Logical *doesRootConnectionExist* ()

Returns true if the Diagram object has a root Connection object associated with the it, else false.

const KeyList& *getKeysOfNotes* ()

Returns the keys of Note objects associated with the Diagram object.

void *getNameAndVersion* (std::string& nameOut, Uint32& versionOut)

Returns the name and version number of the Diagram object in the given arguments.

- const NameVersionPairList& *getNameVersionPairsOfProperties* ()**
Returns the NameVersionPairs of Property objects that are associated with the Diagram object.
- const NameVersionPairList& *getNameVersionPairsOfPropertyGroups* ()**
Returns the NameVersionPairs of PropertyGroup objects that are associated with the Diagram object.
- const Note& *getNote* (const std::string& key)**
Returns the Note object with the given key.
- PropertyPtr *getProperty* (const std::string& name, Uint32 version = 0)**
Returns a reference pointer to the Property object with the given name and version number that is contained by this Diagram object. If only the name is given, it is assumed to be the unique identifier of the Property object.
- PropertyGroupPtr *getPropertyGroup* (const std::string& name, Uint32 version = 0)**
Returns a reference pointer to the PropertyGroup object with the given name and version number that is contained by this Diagram object. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.
- ConnectionPtr *getRootConnection* ()**
Returns a ConnectionPtr to the Connection object, that is the root connection of the Diagram object.
- const SystemPtrList& *getSystemsUsingDiagram* ()**
Returns a SystemPtrList of all System objects that use the Diagram object.
- SystemPtr *getSystemUsingDiagram* (const std::string& name, Uint32 version = 0)**
Returns a SystemPtr to the System object, which is specified by the given name and version, that uses the Diagram object. If only the name is given, it is the unique id of the System object.
- const UniqueIdList& *getUidsOfProperties* ()**
Returns the unique identifies of Property objects that are associated with the Diagram object.
- const UniqueIdList& *getUidsOfPropertyGroups* ()**
Returns the unique identifies of PropertyGroup objects that are associated with the Diagram object.
- const std::string& *getUidOfRootConnection* ()**
The unique id of the Connection object that is associated with the Diagram object as the root connection is returned. An empty string is returned if there is no root connection.
- const UniqueIdList& *getUidsOfSystemsUsingDiagram* ()**
A list of unique ids of the System objects that use the Diagram object.
- void *modifyNote* (const std::string& key, const std::string& comment)**
Modifies the Note object with the given key, that is associated with the Diagram object, by replacing the comment with the given comment.
- void *removeNote* (const std::string& key)**
Removes the Note object with the given key from the list of Note objects associated with the Diagram object.

void *removeRootConnection* ()

Removes the Connection object that is associated with the Diagram object as its root connection.

DiagramPtr

DiagramPtr is a type definition for a reference pointer to a Diagram object.

DiagramPtrList

DiagramPtrList is a type definition for a list of reference pointers to Diagram objects. It is defined as follows:

`std::vector<DiagramPtr>`.

DiagramPtrMap

DiagramPtrMap is a type definition for an associative map between unique identifiers of Diagram objects and their reference pointers. It is defined as follows:

`std::map<std::string, DiagramPtr>`

Factory Classes

The Factory Package contains the class that manages the LEAPS database.

Factory

The purpose of a Factory object is to manage a Leaps Database.

Public Attributes:

UInt32 *numberOfStudies* ()

Number of Study objects created by the Factory.

Public Operations:

FactoryPtr *create* (const std::string& dbName, Int32 maxCmem = 1000000, Int32 maxImem = 5000000, Int32 maxDmem = 2500000)

Creates a reference pointer (FactoryPtr) to a Leaps factory object to manage the given Leaps database name.

StudyPtr *createStudy* (const std::string& name, UInt32 version)

Creates a Study object with the given name and version number and returns a reference pointer (StudyPtr) to the object.

void *destroyStudy* (const std::string& name, UInt32 version = 0)

Destroys the Study object with the given name and version number that is part of Factory object. If only the name is given, it is assumed to be the unique identifier of the Study object.

Logical *doesStudyExist* (const std::string& name, UInt32 version = 0)

Returns true if Study object exists, else false

const NameVersionPairList& *getNameVersionPairsOfStudies* ()

Returns a list of the NameVersionPairs of the Study objects that are a part of the Factory.

StudyPtr *getStudy* (const std::string& name, Uint32 version = 0)

Returns a StudyPtr to the Study object, which is specified by the given name and version, that is part of Factory object. If only the name is given, it is the unique identifier of the Study object.

const UniqueIdList& *getUidsOfStudies* ()

Returns a list of the unique identifiers of the Study objects that are a part of the Factory.

FactoryPtr

FactoryPtr is a type definition for a reference pointer to a Factory object.

FactoryPtrMap

FactoryPtrMap is a type definition for an associative map between the database names of Factory objects and their reference pointers. It is defined as follows:
std::map<std::string, FactoryPtr>

Scenario Classes

Scenario

The purpose of the Scenario Class is to provide the operational situation in which the concepts of the study are required to operate. The operational situation provides the context of how the concepts are to be evaluated and the measures of effectiveness.

Public Attributes:

Name *id* ()

Name object that identifies the Scenario object.

std::string *name* ()

Name of the Scenario object.

Uint32 *numberOfNotes* ()

Number of Note objects associated with the Scenario object.

Uint32 *numberOfProperties* ()

Number of Property objects associated with the Scenario object.

Uint32 *numberOfPropertyGroups* ()

Number of PropertyGroup objects associated with the Scenario object.

std::string& *uniqueId* ()

Unique identifier of the Scenario object.

Uint32 *version* ()

Version number of the Scenario object.

Public Operations:

void *addNote* (const std::string& key, const std::string& comment)

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the Scenario object.

PropertyPtr *createProperty* (const std::string& name, Uint32 version, const PropertyDataPtr& propData)

Creates a Property object with the given name, version, and property data. This object is associated with the Scenario object.

PropertyGroupPtr *createPropertyGroup* (const std::string& name, Uint32 version, const PropertyPtrList& propertyData, const PropertyGroupPtrList& propertyGroupData)

Creates a PropertyGroup object with the given name, version, and a list of reference pointers to PropertyGroup and Property objects. This PropertyGroup object is associated with the Scenario object.

void *debugPrint* ()

Prints the Scenario object's member values to the error file cerr.

void *destroyProperty* (const std::string& name, Uint32 version = 0)

Destroys the Property object with the given name and version that is contained by this Scenario object. If only the name is given, it is assumed to be the unique identifier of the Property object.

void *destroyPropertyGroup* (const std::string& name, Uint32 version = 0)

Destroys the PropertyGroup object with the given name and version that is contained by this Scenario object. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

Logical *doesNoteExist* (const std::string& key)

Returns true if the Note object with the given key is associated with the Scenario object, else false

Logical *doesPropertyExist* (const std::string& name, Uint32 version = 0)

Returns true if the Property object with the given name and version number is associated with the Scenario object, else false. If only the name is given, it is assumed to be the unique identifier of the Property object.

Logical *doesPropertyGroupExist* (const std::string& name, Uint32 version = 0)

Returns true if the PropertyGroup object with the given name and version number is associated with the Scenario object, else false. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

const KeyList& *getKeysOfNotes* ()

Returns the keys of Note objects associated with the Scenario object.

void *getNameAndVersion* (std::string& nameOut, Uint32& versionOut)

Returns the name and version number of the Scenario object in the given arguments.

const NameVersionPairList& *getNameVersionPairsOfProperties* ()

Returns the NameVersionPairs of Property objects that are associated with the Scenario object.

const NameVersionPairList& *getNameVersionPairsOfPropertyGroups* ()

Returns the NameVersionPairs of PropertyGroup objects that are associated with the Scenario object.

const Note& *getNote* (const std::string& key)

Returns the Note object with the given key.

PropertyPtr *getProperty* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the Property object with the given name and version number that is contained by this Scenario object. If only the name is given, it is assumed to be the unique identifier of the Property object.

PropertyGroupPtr *getPropertyGroup* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the PropertyGroup object with the given name and version number that is contained by this Scenario object. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

const UniqueIdList& *getUidsOfProperties* ()

Returns the unique identifies of Property objects that are associated with the Scenario object.

const UniqueIdList& *getUidsOfPropertyGroups* ()

Returns the unique identifies of PropertyGroup objects that are associated with the Scenario object.

void *modifyNote* (const std::string& key, const std::string& comment)

Modifies the Note object with the given key, that is associated with the Scenario object, by replacing the comment with the given comment.

void *removeNote* (const std::string& key)

Removes the Note object with the given key from the list of Note objects associated with the Scenario object.

ScenarioPtr

ScenarioPtr is a type definition for a reference pointer to a Scenario object.

ScenarioPtrList

ScenarioPtrList is a type definition for a list of reference pointers to Scenario objects.. It is defined as follows:
`std::vector<ScenarioPtr>`

ScenarioPtrMap

ScenarioPtrMap is a type definition for an associative map between unique identifiers of Scenario objects and their reference pointers.. It is defined as follows:

`std::map<std::string, ScenarioPtr>`

Study Classes

The Study Package contains classes that compose the Study Class.

Study

The integrated process team (IPT) is created to perform a study. The Study class is used to represent this study. The Study class is composed of properties, property groups, concepts, and scenarios.

Public Attributes:

Name *id* ()

Name object that identifies the Study object.

std::string *name* ()

Name of the Study object.

UInt32 *numberOfConcepts* ()

Number of Concept objects that are a part of the Study object.

UInt32 *numberOfNotes* ()

Number of Note objects associated with the Study object.

UInt32 *numberOfProperties* ()

Number of Property objects associated with the Study object.

UInt32 *numberOfPropertyGroups* ()

Number of PropertyGroup objects associated with the Study object.

UInt32 *numberOfScenarios* ()

Number of Scenario objects that are a part of the Study object.

std::string& *uniqueId* ()

Unique identifier of the Study object.

UInt32 *version* ()

Version number of the Study object.

Public Operations:

void *addNote* (const std::string& key, const std::string& comment)

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the Study object.

ConceptPtr *createConcept* (const std::string& name, UInt32 version)

Creates a Concept object with the given name and version number. This Concept object is made a part of the Study object and a reference pointer (ConceptPtr) is returned.

PropertyPtr *createProperty* (const std::string& name, UInt32 version, const PropertyDataPtr& propData)

Creates a Property object with the given name, version, and property data. This object is associated with the Study object.

PropertyGroupPtr *createPropertyGroup* (const std::string& name, UInt32 version, const PropertyPtrList& propertyData, const PropertyGroupPtrList& propertyGroupData)

Creates a PropertyGroup object with the given name, version, and a list of reference pointers to PropertyGroup and Property objects. This PropertyGroup object is associated with the Study object.

ScenarioPtr *createScenario* (const std::string& name, UInt32 version)

Creates a Scenario object with the given name and version number. This Scenario object is made a part of the Study object and a reference pointer (ScenarioPtr) is returned.

void *destroyConcept* (const std::string& name, UInt32 version = 0)

Destroys the Concept object with the given name and version that is contained by this Study object. If only the name is given, it is assumed to be the unique identifier of the Concept object.

void *destroyProperty* (const std::string& name, UInt32 version = 0)

Destroys the Property object with the given name and version that is contained by this Study object. If only the name is given, it is assumed to be the unique identifier of the Property object.

void *destroyPropertyGroup* (const std::string& name, Uint32 version = 0)
Destroys the PropertyGroup object with the given name and version that is contained by this Study object. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

void *destroyScenario* (const std::string& name, Uint32 version = 0)
Destroys the Scenario object with the given name and version that is contained by this Study object. If only the name is given, it is assumed to be the unique identifier of the Scenario object.

void *debugPrint* ()
Prints the Study object's member values to the error file cerr.

Logical *doesConceptExist* (const std::string& name, Uint32 version = 0)
Returns true if the Concept object with the given name and version number is associated with the Study object, else false. If only the name is given, it is assumed to be the unique identifier of the Concept object.

Logical *doesNoteExist* (const std::string& key)
Returns true if the Note object with the given key is associated with the Study object, else false

Logical *doesPropertyExist* (const std::string& name, Uint32 version = 0)
Returns true if the Property object with the given name and version number is associated with the Study object, else false. If only the name is given, it is assumed to be the unique identifier of the Property object.

Logical *doesPropertyGroupExist* (const std::string& name, Uint32 version = 0)
Returns true if the PropertyGroup object with the given name and version number is associated with the Study object, else false. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

Logical *doesScenarioExist* (const std::string& name, Uint32 version = 0)
Returns true if the Scenario object with the given name and version number is associated with the Study object, else false. If only the name is given, it is assumed to be the unique identifier of the Scenario object.

ConceptPtr *getConcept* (const std::string& name, Uint32 version = 0)
Returns a reference pointer (ConceptPtr) to the Concept object, which is specified by the given name and version, that is part of Study object. If only the name is given, it is assumed to be the unique id of the Concept object.

const KeyList& *getKeysOfNotes* ()
Returns the keys of Note objects associated with the Study object.

void *getNameAndVersion* (std::string& nameOut, Uint32& versionOut)
Returns the name and version number of the Study object in the given arguments.

const NameVersionPairList& *getNameVersionPairsOfConcepts* ()
Returns the NameVersionPairs of Concept objects that are associated with the Study object.

const NameVersionPairList& *getNameVersionPairsOfProperties* ()
Returns the NameVersionPairs of Property objects that are associated with the Study object.

const NameVersionPairList& *getNameVersionPairsOfPropertyGroups* ()
Returns the NameVersionPairs of PropertyGroup objects that are associated with the Study object.

const NameVersionPairList& *getNameVersionPairsOfScenarios* ()

Returns the NameVersionPairList of Scenario objects that are associated with the Study object.

const Note& *getNote* (const std::string& key)

Returns the Note object with the given key.

PropertyPtr *getProperty* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the Property object with the given name and version number that is contained by this Study object. If only the name is given, it is assumed to be the unique identifier of the Property object.

PropertyGroupPtr *getPropertyGroup* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the PropertyGroup object with the given name and version number that is contained by this Study object. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

ScenarioPtr *getScenario* (const std::string& name, Uint32 version = 0)

Returns a reference pointer (ScenarioPtr) to the Scenario object, which is specified by the given name and version, that is part of Study object. If only the name is given, it is assumed to be the unique id of the Scenario object.

const UniqueIdList& *getUidsOfConcepts* ()

Returns the unique identifies of Concept objects that are associated with the Study object.

const UniqueIdList& *getUidsOfProperties* ()

Returns the unique identifies of Property objects that are associated with the Study object.

const UniqueIdList& *getUidsOfPropertyGroups* ()

Returns the unique identifies of PropertyGroup objects that are associated with the Study object.

const UniqueIdList& *getUidsOfScenarios* ()

Returns the unique identifies of Scenario objects that are associated with the Study object.

void *modifyNote* (const std::string& key, const std::string& comment)

Modifies the Note object with the given key, that is associated with the Study object by replacing the comment with the given comment.

void *removeNote* (const std::string& key)

Removes the Note object with the given key from the list of Note objects associated with the Study object.

StudyPtr

StudyPtr is a type definition for a reference pointer to a Study object.

StudyPtrList

StudyPtrList is a type definition for a list of reference pointers to Study objects. It is defined as follows:

std::vector<StudyPtr>.

StudyPtrMap

StudyPtrMap is a type definition for an associative map between unique identifiers of Study objects and their reference pointers. It is defined as follows:
std::map<std::string, StudyPtr>.

System Classes

System

The System class represents those objects that are viewed as systems of a concept. The System class is composed of subsystems and/or components. The system has properties and a component view of the system.

Public Attributes:

Name *id* ()

Name object that identifies the System object.

std::string *name* ()

Name of the System object.

UInt32 *numberOfConnectionsUsingSystem* ()

Number of Connection objects that use this System object.

UInt32 *numberOfDiagrams* ()

Number of Diagram objects that are a part of the System object.

UInt32 *numberOfNotes* ()

Number of Note objects associated with the System object.

UInt32 *numberOfProperties* ()

Number of Property objects associated with the System object.

UInt32 *numberOfPropertyGroups* ()

Number of PropertyGroup objects associated with the System object.

UInt32 *numberOfComponents* ()

Number of Component objects that compose the System object.

UInt32 *numberOfSystems* ()

Number of System objects that compose the System object.

UInt32 *numberOfSystemsUsingSystem* ()

Number of System objects that use this System object.

std::string& *uniqueId* ()

Unique identifier of the System object.

UInt32 *version* ()

Version number of the System object.

Public Operations:

void *addComponent* (const ComponentPtr& componentToAdd)

Adds the association of the given Component object with the System object.

- void *addComponents* (const ComponentPtrList& componentListToAdd)**
Adds the association of the given list of Component objects with the System object.
- void *addDiagram* (const DiagramPtr& diagramToAdd)**
Adds the association of the given Diagram object with the System object.
- void *addDiagrams* (const DiagramPtrList& diagramListToAdd)**
Adds the association of the given list of Diagram objects with the System object.
- void *addNote* (const std::string& key, const std::string& comment)**
Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the System object.
- void *addSystem* (const SystemPtr& systemToAdd)**
Adds the association of the given System object with the System object.
- void *addSystems* (const SystemPtrList& systemListToAdd)**
Adds the association of the given list of System objects with the System object.
- PropertyPtr *createProperty* (const std::string& name, Uint32 version, const PropertyDataPtr& propData)**
Creates a Property object with the given name, version, and property data. This object is associated with the System object.
- PropertyGroupPtr *createPropertyGroup* (const std::string& name, Uint32 version, const PropertyPtrList& propertyData, const PropertyGroupPtrList& propertyGroupData)**
Creates a PropertyGroup object with the given name, version, and a list of reference pointers to PropertyGroup and Property objects. This PropertyGroup object is associated with the System object.
- void *debugPrint* ()**
Prints the System object's member values to the error file cerr.
- void *destroyProperty* (const std::string& name, Uint32 version = 0)**
Destroys the Property object with the given name and version that is contained by this System object. If only the name is given, it is assumed to be the unique identifier of the Property object.
- void *destroyPropertyGroup* (const std::string& name, Uint32 version = 0)**
Destroys the PropertyGroup object with the given name and version that is contained by this System object. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.
- Logical *doesNoteExist* (const std::string& key)**
Returns true if the Note object with the given key is associated with the System object, else false
- Logical *doesPropertyExist* (const std::string& name, Uint32 version = 0)**
Returns true if the Property object with the given name and version number is associated with the System object, else false. If only the name is given, it is assumed to be the unique identifier of the Property object.
- Logical *doesPropertyGroupExist* (const std::string& name, Uint32 version = 0)**
Returns true if the PropertyGroup object with the given name and version number is associated with the System object, else false. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

ComponentPtr *GetComponent* (const std::string& name, Uint32 version = 0)

Returns a ComponentPtr to the Component object, which is specified by the given name and version, that is part of System object. If only the name is given, it is the unique id of the Component object.

const ComponentPtrList& *getComponents* ()

A list of reference pointers (ComponentPtrList) to all Component objects that compose the System object is returned.

const ConnectionPtrList& *getConnectionsUsingSystem* ()

Returns a ConnectionPtrList of all Connection objects that use this System object.

ConnectionPtr *getConnectionUsingSystem* (const std::string& name, Uint32 version = 0)

Returns a ConnectionPtr to the Connection object, given the specified name and version, that uses this System object. If only the name is given, it is assumed to be the unique id of the Connection object.

DiagramPtr *getDiagram* (const std::string& name, Uint32 version = 0)

Returns a DiagramPtr to the Diagram object, which is specified by the given name and version, that is associated with the System object. If only the name is given, it is assumed to be the unique id of the Diagram object.

const DiagramPtrList& *getDiagrams* ()

Returns a DiagramPtrList of all Diagram objects that are associated with the System object.

const KeyList& *getKeysOfNotes* ()

Returns the keys of Note objects associated with the System object.

void *getNameAndVersion* (std::string& nameOut, Uint32& versionOut)

Returns the name and version number of the System object in the given arguments.

const NameVersionPairList& *getNameVersionPairsOfProperties* ()

Returns the NameVersionPairs of Property objects that are associated with the System object.

const NameVersionPairList& *getNameVersionPairsOfPropertyGroups* ()

Returns the NameVersionPairs of PropertyGroup objects that are associated with the System object.

const Note& *getNote* (const std::string& key)

Returns the Note object with the given key.

PropertyPtr *getProperty* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the Property object with the given name and version number that is contained by this System object. If only the name is given, it is assumed to be the unique identifier of the Property object.

PropertyGroupPtr *getPropertyGroup* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the PropertyGroup object with the given name and version number that is contained by this System object. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

SystemPtr *getSystem* (const std::string& name, Uint32 version = 0)

Returns a SystemPtr to the System object, which is specified by the given name and version, that is part of System object. If only the name is given, it is assumed to be the unique id of the System object.

const SystemPtrList& *getSystems* ()

Returns a SystemPtrList of all System objects that compose the System object.

const SystemPtrList& *getSystemsUsingSystem* ()

Returns a SystemPtrList of all System objects that use this System object.

SystemPtr *getSystemUsingSystem* (const std::string& name, Uint32 version = 0)

Returns a SystemPtr to the System object, given the specified name and version, that is used by the System object. If only the name is given, it is assumed to be the unique id of the System object.

const UniqueIdList& *getUidsOfComponents* ()

A list of unique ids of the components that compose the system is returned.

const UniqueIdList& *getUidsOfConnectionsUsingSystem* ()

A list of unique ids of the Connection objects that use this System object.

const UniqueIdList& *getUidsOfDiagrams* ()

A list of unique ids of the diagrams that are associated with the system.

const UniqueIdList& *getUidsOfProperties* ()

Returns the unique identifies of Property objects that are associated with the System object.

const UniqueIdList& *getUidsOfPropertyGroups* ()

Returns the unique identifies of PropertyGroup objects that are associated with the System object.

const UniqueIdList& *getUidsOfSystems* ()

A list of unique ids of the systems that compose the system is returned.

const UniqueIdList& *getUidsOfSystemsUsingSystem* ()

A list of unique ids of the systems that use this system.

void *modifyNote* (const std::string& key, const std::string& comment)

Modifies the Note object with the given key, that is associated with the System object by replacing the comment with the given comment.

void *removeComponent* (const ComponentPtr& componentToRemove)

Removes the association of the given Component object with the System object.

void *removeComponents* (const ComponentPtrList& componentListToRemove)

Removes the association of the given list of Component objects with the System object.

void *removeDiagram* (const DiagramPtr& diagramToRemove)

Removes the association of the given Diagram object with the System object.

void *removeDiagrams* (const DiagramPtrList& diagramListToRemove)

Removes the association of the given list of Diagram objects with the System object.

void *removeNote* (const std::string& key)

Removes the Note object with the given key from the list of Note objects associated with the System object.

void *removeSystem* (const SystemPtr& systemToRemove)

Removes the association of the given System object with the System object.

void *removeSystems* (const SystemPtrList& systemListToRemove)

Removes the association of the given list of System objects with the System object.

SystemPtr

SystemPtr is a type definition for a reference pointer to a System object.

SystemPtrList

SystemPtrList is a type definition for a list of reference pointers to System objects. It is defined as follows:

std::vector<SystemPtr>.

SystemPtrMap

SystemPtrMap is a type definition for an associative map between unique identifiers of System objects and their reference pointers. It is defined as follows:

std::map<std::string, SystemPtr>.

LEAPS GEOMETRY OBJECT STRUCTURE (GOBS)

Gobs Package contains classes that implement the Geometry Object Structure (GOBS).

CoEdge Classes

CoEdge

A CoEdge object defines the relationship between two or more Edges. The relationship is defined where a CoEdge knows 1) all Edges that compose it; and 2) the equivalent Cartesian location on all member Edges, expressed as a parametric location on the underlying Pcurve, for any location on the CoEdge. The CoEdge is used to allow traversal across Surfaces or Faces and defines explicitly an association between two or more Surfaces or Faces.

Derived from Spline

Public Attributes:

Logical *consideredStraight* (const Real64 tolerance = 0.001)

Returns true if the CoEdge object is straight within the given tolerance, else false.

CartesianLocation *endLocation* ()

The cartesian location (x, y, z) that ends the CoEdge object.

Name *id* ()

Name object that identifies the CoEdge object.

std::string *name* ()

Name of the CoEdge object.

UInt32 *numberOfEdges* ()

Number of Edge objects that compose the CoEdge object.

UInt32 *numberOfNotes* ()

Number of Note objects associated with the CoEdge object.

CartesianLocation *startLocation* ()

The cartesian location (x, y, z) that starts the CoEdge object.

std::string& *uniqueId* ()

Unique identifier of the CoEdge object.

UInt32 *version* ()

Version number of the CoEdge object.

Public Operations:

void *addNote* (const std::string& key, const std::string& comment)

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the CoEdge object.

void *debugPrint* ()

Prints the CoEdge object's member values to the error file cerr.

Logical *doesNoteExist* (const std::string& key)

Returns true if the Note object with the given key is associated with the CoEdge object, else false

std::pair<Real64, std::vector<PcurveLocation> > *evlForCoEdgeLoc* (const Real64 s)

Evaluate CoEdge for PcurveLocations at the given s. A STL pair is returned where the first element is the parametric value s at which the CoEdge was evaluated and the second element is a STL vector of PcurveLocation objects.

const std::vector<std::vector<PcurveLocation> >& *evlAtEqualParametric* (UInt32 numLoc = 2)

Evaluate CoEdge for surface (u,v) and (x, y, z) given a number of points. A STL vector of STL vectors of PcurveLocation is returned. A PcurveLocation is defined as a STL vector of data where the curve domain parameter s, is at index 0, the surface domain parameter u, is at index 1, the surface domain parameter v, is at index 2, the surface range parameter x is at index 3, the surface range parameter y is at index 4, and the surface range parameter z is at index 5.

EdgePtr *getEdge* (const std::string& name, UInt32 version = 0)

Returns a reference pointer (EdgePtr) to the Edge object with the specified name and version that composes the CoEdge object. If only the name is given, it is assumed to be the unique identifier of the Edge object.

const EdgePtrList& *getEdges* ()

Returns a list (EdgePtrList) of reference pointers to Edge objects that compose the CoEdge object.

const KeyList& *getKeysOfNotes* ()

Returns the keys of Note objects associated with the CoEdge object.

void *getNameAndVersion* (std::string& nameOut, UInt32& versionOut)

Returns the name and version number of the CoEdge object in the given arguments.

const Note& *getNote* (const std::string& key)

Returns the Note object with the given key.

const PcurvePtrList& *getPcurves* ()

Returns a list (PcurvePtrList) of reference pointers to Pcurve objects that are a part of the CoEdge object.

const SurfacePtrList& *getSurfaces* ()

Returns a list (SurfacePtrList) of reference pointers to Surface objects that are a part of the CoEdge object.

const UniqueldList& *getUidsOfEdges* ()

Returns a list (UniqueldList) of unique identifiers of Edge objects that compose the CoEdge object.

const UniqueldList& *getUidsOfPcurves* ()

Returns a list (UniqueldList) of unique identifiers of Pcurve objects that are associated with the CoEdge object.

const UniqueldList& *getUidsOfSurfaces* ()

Returns a list (UniqueldList) of unique identifiers of Surface objects that are part of the CoEdge object.

void *modifyNote* (const std::string& key, const std::string& comment)

Modifies the Note object with the given key, that is associated with the CoEdge object by replacing the comment with the given comment.

void *removeEdge* (EdgePtr& edgeToRemove)

Removes the association of the given Edge object with the CoEdge object.

void *removeNote* (const std::string& key)

Removes the Note object with the given key from the list of Note objects associated with the CoEdge object.

CoEdgePtr

CoEdgePtr is a type definition for a reference pointer to a CoEdge object.

CoEdgePtrList

CoEdgePtrList is a type definition for a list of reference pointers to CoEdge objects.

CoEdgePtrMap

CoEdgePtrMap is a type definition for an associative map between unique identifiers of CoEdge objects and their reference pointers.

CommonView Classes

CommonView

The CommonView class provides a logical view of the structure. The CommonView class is composed of TopologicalView objects and/or other CommonView objects.

Public Attributes:

Name *id* ()

Name object that identifies the CommonView object.

std::string *name* ()

Name of the CommonView object.

UInt32 *numberOfCommonViews* ()

Number of CommonView objects that compose the CommonView object.

UInt32 *numberOfCommonViewsUsingCommonView* ()

Number of CommonView objects that use the CommonView object.

UInt32 *numberOfNotes* ()

Number of Note objects associated with the CommonView object.

UInt32 *numberOfMaterials* ()

Number of Material objects associated with the CommonView object.

UInt32 *numberOfMaterialGroups* ()

Number of MaterialGroup objects associated with the CommonView object.

UInt32 *numberOfProperties* ()

Number of Property objects associated with the CommonView object.

UInt32 *numberOfPropertyGroups* ()

Number of PropertyGroup objects associated with the CommonView object.

UInt32 *numberOfTopologicalViews* ()

Number of TopologicalView objects that compose the CommonView object.

std::string *uniqueId* ()

Unique identifier of the CommonView object.

UInt32 *version* ()

Version number of the CommonView object.

Public Operations:

void *addCommonView* (const CommonViewPtr& commonViewToAdd)

Adds the association of the given CommonView object with the CommonView object.

void *addNote* (const std::string& key, const std::string& comment)

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the CommonView object.

void *addTopologicalView* (const TopologicalViewPtr& topologicalViewToAdd)

Adds the association of the given TopologicalView object with the CommonView object.

MaterialPtr *createMaterial* (const std::string& name, UInt32 version)

Creates a Material object with the given name and version. This object is associated with the CommonView object.

MaterialGroupPtr *createMaterialGroup* (const std::string& name, UInt32 version, const MaterialPtrList& materialData, const MaterialGroupPtrList& materialGroupData)

Creates a MaterialGroup object with the given name, version, and a list of reference pointers to MaterialGroup and Material objects. This MaterialGroup object is associated with the CommonView object.

PropertyPtr *createProperty* (const std::string& name, Uint32 version, const PropertyDataPtr& propData)

Creates a Property object with the given name, version, and property data. This object is associated with the CommonView object.

PropertyGroupPtr *createPropertyGroup* (const std::string& name, Uint32 version, const PropertyPtrList& propertyData, const PropertyGroupPtrList& propertyGroupData)

Creates a PropertyGroup object with the given name, version, and a list of reference pointers to PropertyGroup and Property objects. This PropertyGroup object is associated with the CommonView object.

void *debugPrint* ()

Prints the CommonView object's member values to the error file cerr.

void *destroyMaterial* (const std::string& name, Uint32 version = 0)

Destroys the Material object with the given name and version that is contained by this CommonView object. If only the name is given, it is assumed to be the unique identifier of the Material object.

void *destroyMaterialGroup* (const std::string& name, Uint32 version = 0)

Destroys the MaterialGroup object with the given name and version that is contained by this CommonView object. If only the name is given, it is the unique identifier of the MaterialGroup object.

void *destroyProperty* (const std::string& name, Uint32 version = 0)

Destroys the Property object with the given name and version that is contained by this CommonView object. If only the name is given, it is assumed to be the unique identifier of the Property object.

void *destroyPropertyGroup* (const std::string& name, Uint32 version = 0)

Destroys the PropertyGroup object with the given name and version that is contained by this CommonView object. If only the name is given, it is the unique identifier of the PropertyGroup object.

Logical *doesMaterialExist* (const std::string& name, Uint32 version = 0)

Returns true if the Material object with the given name and version number is associated with the CommonView object, else false. If only the name is given, it is the unique identifier of the Material object.

Logical *doesMaterialGroupExist* (const std::string& name, Uint32 version = 0)

Returns true if the MaterialGroup object with the given name and version number is associated with the CommonView object, else false. If only the name is given, it is the unique identifier of the MaterialGroup object.

Logical *doesNoteExist* (const std::string& key)

Returns true if the Note object with the given key is associated with the CommonView object, else false

Logical *doesPropertyExist* (const std::string& name, Uint32 version = 0)

Returns true if the Property object with the given name and version number is associated with the CommonView object, else false. If only the name is given, it is the unique identifier of the Property object.

Logical *doesPropertyGroupExist* (const std::string& name, Uint32 version = 0)

Returns true if the PropertyGroup object with the given name and version number is associated with the CommonView object, else false. If only the name is given, it is the unique identifier of the PropertyGroup object.

CommonViewPtr *getCommonView* (const std::string& name, Uint32 version = 0)

Returns a CommonViewPtr to the CommonView object, which is specified by the given name and version, that is part of CommonView object. If only the name is given, it is the unique id of the CommonView object.

const CommonViewPtrList& *getCommonViews* ()

Returns a CommonViewPtrList of all CommonView objects that compose the CommonView object.

const CommonViewPtrList& *getCommonViewsUsingCommonView* ()

Returns a CommonViewPtrList of all CommonView objects that the CommonView object is a part of.

CommonViewPtr *getCommonViewUsingCommonView* (const std::string& name, Uint32 version = 0)

Returns a CommonViewPtr to the CommonView object, given the specified name and version, that the CommonView object is a part of. If only the name is given, it is assumed to be the unique id of the CommonView object.

const KeyList& *getKeysOfNotes* ()

Returns the keys of Note objects associated with the CommonView object.

MaterialPtr *getMaterial* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the Material object with the given name and version number that is contained by this CommonView object. If only the name is given, it is the unique identifier of the Material object.

MaterialGroupPtr *getMaterialGroup* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the MaterialGroup object with the given name and version number that is contained by this CommonView object. If only the name is given, it is assumed to be the unique identifier of the MaterialGroup object.

void *getNameAndVersion* (std::string& nameOut, Uint32& versionOut)

Returns the name and version number of the CommonView object in the given arguments.

const NameVersionPairList& *getNameVersionPairsOfMaterials* ()

Returns the NameVersionPairs of Material objects that are associated with the CommonView object.

const NameVersionPairList& *getNameVersionPairsOfMaterialGroups* ()

Returns the NameVersionPairs of MaterialGroup objects that are associated with the CommonView object.

const NameVersionPairList& *getNameVersionPairsOfProperties* ()

Returns the NameVersionPairs of Property objects that are associated with the CommonView object.

const NameVersionPairList& *getNameVersionPairsOfPropertyGroups* ()

Returns the NameVersionPairs of PropertyGroup objects that are associated with the CommonView object.

const Note& *getNote* (const std::string& key)

Returns the Note object with the given key.

PropertyPtr *getProperty* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the Property object with the given name and version number that is contained by this CommonView object. If only the name is given, it is assumed to be the unique identifier of the Property object.

PropertyGroupPtr *getPropertyGroup* (const std::string& name, Uint32 version = 0)
Returns a reference pointer to the PropertyGroup object with the given name and version number that is contained by this CommonView object. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

TopologicalViewPtr *getTopologicalView* (const std::string& name, Uint32 version = 0)
Returns a TopologicalViewPtr to the TopologicalView object, which is specified by the given name and version, that is part of CommonView object. If only the name is given, it is the unique id of the TopologicalView object.

const TopologicalViewPtrList& *getTopologicalViews* ()
Returns a TopologicalViewPtrList of all TopologicalView objects that compose the CommonView object.

const UniqueIdList& *getUidsOfCommonViews* ()
A list of unique ids of the common views that are compose the common view.

const UniqueIdList& *getUidsOfCommonViewsUsingCommonView* ()
A list of unique ids of the common views that the common view is a part of.

const UniqueIdList& *getUidsOfMaterials* ()
Returns the unique identifies of Material objects that are associated with the CommonView object.

const UniqueIdList& *getUidsOfMaterialGroups* ()
Returns the unique identifies of MaterialGroup objects that are associated with the CommonView object.

const UniqueIdList& *getUidsOfProperties* ()
Returns the unique identifies of Property objects that are associated with the CommonView object.

const UniqueIdList& *getUidsOfPropertyGroups* ()
Returns the unique identifies of PropertyGroup objects that are associated with the CommonView object.

const UniqueIdList& *getUidsOfTopologicalViews* ()
A list of unique ids of the topological views that are compose the common view.

void *modifyNote* (const std::string& key, const std::string& comment)
Modifies the Note object with the given key, that is associated with the CommonView object by replacing the comment with the given comment.

void *removeCommonView* (const CommonViewPtr& commonViewToRemove)
Removes the association of a CommonView object with the CommonView object.

void *removeNote* (const std::string& key)
Removes the Note object with the given key from the list of Note objects associated with the CommonView object.

void *removeTopologicalView* (const TopologicalViewPtr& topologicalViewToRemove)
Removes the association of a TopologicalView object with the CommonView object.

CommonViewPtr

CommonViewPtr is a type definition for a reference pointer to a CommonView object.

CommonViewPtrList

CommonViewPtrList is a type definition for a list of reference pointers to CommonView objects.

CommonViewPtrMap

CommonViewPtrMap is a type definition for an associative map between unique identifiers of CommonView objects and their reference pointers.

CoPoint Classes

CoPoint

A CoPoint defines the CartesianLocation equivalent for a list of Ppoint objects.

Public Attributes:

Name *id* ()

Name object that identifies the CoPoint object.

CartesianLocation *location* ()

CartesianLocation (x,y,z) which specifies the location of the CoPoint object in model space.

std::string *name* ()

Name of the CoPoint object.

UInt32 *numberOfNotes* ()

Number of Note objects associated with the CoPoint object.

UInt32 *numberOfPpoints* ()

Number of Ppoint objects that compose the CoPoint object.

std::string& *uniqueId* ()

Unique identifier of the CoPoint object.

UInt32 *version* ()

Version number of the CoPoint object.

Public Operations:

void *addNote* (const std::string& key, const std::string& comment)

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the CoPoint object.

void *addPpoint* (const PpointPtr& ppointToAdd)

Adds the association of the given Ppoint object with the CoPoint object.

void *debugPrint* ()

Prints the CoPoint object's member values to the error file cerr.

Logical *doesNoteExist* (const std::string& key)

Returns true if the Note object with the given key is associated with the CoPoint object, else false

const KeyList& *getKeysOfNotes* ()

Returns the keys of Note objects associated with the CoPoint object.

void *getNameAndVersion* (std::string& nameOut, Uint32& versionOut)

Returns the name and version number of the CoPoint object in the given arguments.

const Note& *getNote* (const std::string& key)

Returns the Note object with the given key.

PpointPtr *getPpoint* (const std::string& name, Uint32 version = 0)

Returns a reference pointer (PpointPtr) to the Ppoint object, which is specified by the given name and version, that is part of CoPoint object. If only the name is given, it is assumed the unique identifier of the Ppoint object.

const PpointPtrList& *getPpoints* ()

Returns a list (PpointPtrList) of reference pointers to Ppoint objects that compose the CoPoint object

const UniqueldList& *getUidsOfPpoints* ()

Returns a list (UniqueldList) of the unique identifiers to the Ppoint objects that compose the Ppoint object.

void *modifyNote* (const std::string& key, const std::string& comment)

Modifies the Note object with the given key, that is associated with the CoPoint object by replacing the comment with the given comment.

void *removeNote* (const std::string& key)

Removes the Note object with the given key from the list of Note objects associated with the CoPoint object.

void *removePpoint* (const PpointPtr& ppointToRemove)

Removes the association of the given Ppoint object with the CoPoint object.

CoPointPtr

CoPointPtr is a type definition for a reference pointer to a CoPoint object.

CoPointPtrList

CoPointPtrList is a type definition for a list of CoPoint objects.

CoPointPtrMap

CoPointPtrMap is a type definition for an associative map between unique identifiers of CoPoint objects and their reference pointers.

Edge Classes

Edge

An Edge defines a region or segment of a Pcurve. The collection of contiguous Edges are used for composing paths, loops, or topological boundaries.

Public Attributes:

Logical *consideredStraight* (Real64 tolerance = 0.001)

Returns true if the Edge object is straight within the given tolerance, else false.

Name *id* ()

Name object that identifies the Edge object.

Real64 *length3D* ()

Length of Edge object in 3D model space.

std::string *name* ()

Name of the Edge object.

UInt32 *numberOfEdgeLoopsUsingEdge* ()

Number of EdgeLoop objects that use the Edge object.

UInt32 *numberOfNotes* ()

Number of Note objects associated with the Edge object.

EdgePtr *opposite* ()

Edge object that is in the opposite direction of the Edge object.

std::string& *uniqueId* ()

Unique identifier of the Edge object.

UInt32 *version* ()

Version number of the Edge object.

Public Operations:

void *addNote* (const std::string& key, const std::string& comment)

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the Edge object.

void *debugPrint* ()

Prints the Edge object's member values to the error file cerr.

Logical *doesNoteExist* (const std::string& key)

Returns true if the Note object with the given key is associated with the Edge object, else false

const std::vector<PcurveLocation>& *evlAtEqualArc* (UInt32 numLoc = 2)

Evaluate an Edge for surface (u,v) and (x, y, z) at equal arclengths given a number of points. A STL vector of PcurveLocations is returned.

const std::vector<PcurveLocation>& *evlAtEqualParametric* (UInt32 numLoc = 2)

Evaluate Edge for surface (u,v) and (x, y, z) at equal parametric lengths given a number of points. A STL vector of PcurveLocation is returned.

const CartesianLocation *evlForCartesianLoc* (Real64 s)

Evaluate an Edge object for (x, y, z) given (s). A CartesianLocation object is returned.

SplineData *evlForCartesianSpline* ()

Evaluates the Edge for a cartesian spline. The domain (s) of the spline is parameterized from 0 to 1. The range of the spline is (x, y, z). The spline is returned as a SplineData object.

const PcurveLocation *evlForClosestPcrvLoc* (const Real64List& cartesianCoords, Real64 initialGuess, Real64 tolerance, Real64& distance, Int32& numOfIterations, Int32& convergenceInfo)

For a given cartesian point (x, y, z), finds the closest location on the Edge object and returns it as a PcurveLocation object.

SplineData *evlForParametricSpline* ()

Evaluates the Edge for a parametric spline. The domain (s) of the spline is parameterized from 0 to 1. The range of the spline is (u, v). The spline is returned as a SplineData object.

const PcurveLocation *evlForPcurveLoc* (Real64 s)

Evaluate Edge for surface (u,v) and (x, y, z) given (s). A PcurveLocation object is returned.

CoEdgePtr *getCoEdge* ()

Returns a reference pointer (CoEdgePtr) to the CoEdge object that uses the Edge object. If no CoEdge object is used by the Edge object, an invalid reference pointer is returned.

EdgeLoopPtr *getEdgeLoopUsingEdge* (const std::string& name, Uint32 version = 0)

Returns a reference pointer (EdgeLoopPtr) to the EdgeLoop object, given the specified name and version, that uses the Edge object. If only the name is given, it is assumed to be the unique identifier of the EdgeLoop object.

const EdgeLoopPtrList& *getEdgeLoopsUsingEdge* ()

Returns a list (EdgeLoopPtrList) of reference pointers of the EdgeLoop objects that use the Edge object.

PpointPtr *getEndPoint* ()

Returns a reference pointer (PpointPtr) to the Ppoint object that ends the Edge object.

const KeyList& *getKeysOfNotes* ()

Returns the keys of Note objects associated with the Edge object.

void *getNameAndVersion* (std::string& nameOut, Uint32& versionOut)

Returns the name and version number of the Edge object in the given arguments.

const Note& *getNote* (const std::string& key)

Returns the Note object with the given key.

PcurvePtr *getPcurve* ()

Returns a reference pointer (PcurvePtr) to the Pcurve object that the Edge object uses.

SurfacePtr *getSurface* ()

Returns a reference pointer (SurfacePtr) to the Surface object that the Edge object uses.

PpointPtr *getStartPoint* ()

Returns a reference pointer (PpointPtr) to the Ppoint object that starts the Edge object.

const std::string& *getUidOfCoEdge* ()

Returns the unique identifier of the CoEdge object that uses the Edge object. An empty string is returned if no CoEdge object uses the Edge object.

const std::string& *getUidOfEndPoint ()*

Returns the unique identifier to the Ppoint object that ends the Edge object.

const UniqueIdList& *getUidsOfEdgeLoopsUsingEdge ()*

Returns a list (UniqueIdList) of unique identifiers of EdgeLoop objects that use the Edge object.

const std::string& *getUidOfStartPoint ()*

Returns the unique identifier to the Ppoint object that starts the Edge object.

void *modifyNote (const std::string& key, const std::string& comment)*

Modifies the Note object with the given key, that is associated with the Edge object by replacing the comment with the given comment.

void *removeNote (const std::string& key)*

Removes the Note object with the given key from the list of Note objects associated with the Edge object.

EdgePtr

EdgePtr is a type definition for a reference pointer to an Edge object.

EdgePtrList

EdgePtrList is a type definition for a list of Edge objects.

EdgePtrMap

EdgePtrMap is a type definition for an associative map between unique identifiers of Edge objects and their reference pointers.

EdgeLoop Classes

EdgeLoop

An EdgeLoop is a set of connected Edge objects that form a closed loop that is not self intersecting. This loop is also oriented.

Public Attributes:

Name *id ()*

Name object that identifies the EdgeLoop object.

std::string *name ()*

Name of the EdgeLoop object.

UInt32 *numberOfEdges ()*

Number of Edge objects that compose the EdgeLoop object.

UInt32 *numberOfFacesUsingEdgeLoop ()*

Number of Face objects that use the EdgeLoop object.

UInt32 *numberOfNotes ()*

Number of Note objects associated with the EdgeLoop object.

EdgeLoopPtr *opposite ()*

EdgeLoop object that is in the opposite direction of the EdgeLoop object.

OrientationEnum *orientation* ()

Orientation (clockwise or counterclockwise) of the EdgeLoop object.

std::string& *uniqueId* ()

Unique identifier of the EdgeLoop object.

UInt32 *version* ()

Version number of the EdgeLoop object.

Public Operations:

void *addNote* (const std::string& key, const std::string& comment)

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the EdgeLoop object.

void *debugPrint* ()

Prints the EdgeLoop object's member values to the error file cerr.

Logical *doesNoteExist* (const std::string& key)

Returns true if the Note object with the given key is associated with the EdgeLoop object, else false

const std::vector<SplineData>& *evlForCartesianLoop* ()

Evaluates the EdgeLoop for a loop that is composed of splines. Each spline is represents an Edge which is part of the EdgeLoop. The domain (s) of each spline is parameterized from 0 to 1. The range of each spline is (x, y, z). The splines are returned in a STL vector of SplineData objects.

const std::vector<SplineData>& *evlForParametricLoop* ()

Evaluates the EdgeLoop for a loop that is composed of splines. Each spline is represents an Edge which is part of the EdgeLoop. The domain (s) of each spline is parameterized from 0 to 1. The range of each spline is (u, v). The splines are returned in a STL vector of SplineData objects.

EdgePtr *getEdge* (const std::string& name, UInt32 version = 0)

Returns a reference pointer (EdgePtr) to the Edge object, which is specified by the given name and version, that is part of the EdgeLoop object. If only the name is given, it is assumed the unique identifier of the Edge object.

const EdgePtrList& *getEdges* ()

Returns a list (EdgePtrList) of reference pointers to Edge objects that compose the EdgeLoop object

const FacePtrList& *getFacesUsingEdgeLoop* ()

Returns a list (FacePtrList) of reference pointers to Face objects that use the EdgeLoop object.

FacePtr *getFaceUsingEdgeLoop* (const std::string& name, UInt32 version = 0)

Returns a reference pointer (FacePtr) to the Face object, given the specified name and version, that uses the EdgeLoop object. If only the name is given, it is the unique identifier of the Face object.

const KeyList& *getKeysOfNotes* ()

Returns the keys of Note objects associated with the EdgeLoop object.

void *getNameAndVersion* (std::string& nameOut, UInt32& versionOut)

Returns the name and version number of the EdgeLoop object in the given arguments.

const Note& *getNote* (const std::string& key)

Returns the Note object with the given key.

const UniqueIdList& *getUidsOfEdges* ()

Returns a list (UniqueIdList) of the unique identifiers to the Edge objects that compose the EdgeLoop object.

const UniqueIdList& *getUidsOfFacesUsingEdgeLoop* ()

Returns a list (UniqueIdList) of the unique identifiers to the Face objects that use the EdgeLoop object.

void *modifyNote* (const std::string& key, const std::string& comment)

Modifies the Note object with the given key, that is associated with the EdgeLoop object, by replacing the comment with the given comment.

void *removeNote* (const std::string& key)

Removes the Note object with the given key from the list of Note objects associated with the EdgeLoop object.

EdgeLoopPtr

EdgeLoopPtr is a type definition for a reference pointer to an EdgeLoop object.

EdgeLoopPtrList

EdgeLoopPtrList is a type definition for a list of reference pointers to EdgeLoop objects.

EdgeLoopPtrMap

EdgeLoopPtrMap is a type definition for an associative map between unique identifiers of EdgeLoop objects and their reference pointers.

Face Classes

Face

Face represents a region of a surface as a trimmed NURBS surface..

Public Attributes:

Real64 *area* ()

Surface area of the Face object.

Logical *consideredFlat* (Real64 tolerance = 0.001)

Returns true if the Face object is flat within the given tolerance, else false.

Name *id* ()

Name object that identifies the Face object.

std::string *name* ()

Name of the Face object.

UInt32 *numberOfInnerLoops* ()

Number of EdgeLoop objects that are holes in the Face object.

UInt32 *numberOfNotes* ()

Number of Note objects associated with the Face object.

UInt32 *numberOfOrientedClosedShellsUsingFace* ()

Number of OrientedClosedShell objects that use the Face object.

FacePtr *opposite* ()

Face object that has the opposite normal of the Face object.

OrientationEnum *orientation* ()

Orientation (inward or outward) of the normal in relation to the Surface object that the Face object is a part.

std::string& *uniqueId* ()

Unique identifier of the Face object.

UInt32 *version* ()

Version number of the Face object.

Public Operations:

void *addNote* (const std::string& key, const std::string& comment)

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the Face object.

void *debugPrint* ()

Prints the Face object's member values to the error file cerr.

Logical *doesNoteExist* (const std::string& key)

Returns true if the Note object with the given key is associated with the Face object, else false

Logical *doesTopologicalViewExist* ()

Returns true if the Face object is also a TopologicalView object, else false.

EdgeLoopPtr *getInnerLoop* (const std::string& name, UInt32 version = 0)

Returns an EdgeLoopPtr to the EdgeLoop object, given the specified name and version, that represents a hole in the Face object. If only the name is given, it is assumed the unique identifier of the EdgeLoop object.

const EdgeLoopPtrList& *getInnerLoops* ()

Returns a list (EdgeLoopPtrList) of all EdgeLoop objects that represent holes (i.e. inner loops) in the Face object.

const KeyList& *getKeysOfNotes* ()

Returns the keys of Note objects associated with the Face object.

void *getNameAndVersion* (std::string& nameOut, UInt32& versionOut)

Returns the name and version number of the Face object in the given arguments.

const Note& *getNote* (const std::string& key)

Returns the Note object with the given key.

const OrientedClosedShellPtrList& *getOrientedClosedShellsUsingFace* ()

Returns a list (OrientedClosedShellPtrList) of reference pointers to the OrientedClosedShell objects that use the Face object.

OrientedClosedShellPtr *getOrientedClosedShellUsingFace* (const std::string& name, UInt32 version = 0)

Returns a reference pointer (OrientedClosedShellPtr) to the OrientedClosedShell object, given the specified name and version, that uses the Face object. If only the name is given, it is the unique identifier of the OrientedClosedShell object.

EdgeLoopPtr *getOuterLoop* ()

Returns a reference pointer (EdgeLoopPtr) to the EdgeLoop object that bounds the Face object.

SurfacePtr *getSurface* ()

Returns a reference pointer (SurfacePtr) to the Surface object that the Face object is mapped to.

TopologicalViewPtr *getTopologicalView* ()

Returns a reference pointer (TopologicalViewPtr) to the TopologicalView object that is represented by the Face object. If no TopologicalView object is represented by the Face object an invalid reference pointer is returned.

const UniqueldList& *getUidsOfInnerLoops* ()

Returns a list (UniqueldList) of unique identifiers of EdgeLoop objects that represent the holes in the Face object.

const UniqueldList& *getUidsOfOrientedClosedShellsUsingFace* ()

Returns a list (UniqueldList) of unique identifiers of OrientedClosedShell objects that use the Face object..

const std::string& *getUidOfOuterLoop* ()

Returns the unique identifier of the EdgeLoop object that bounds the Face object

const std::string& *getUidOfTopologicalView* ()

Returns the unique identifier of the TopologicalView object that the Face object represents or an empty string if the Face object does not represent a TopologicalView object.

void *modifyNote* (const std::string& key, const std::string& comment)

Modifies the Note object with the given key, that is associated with the Face object, by replacing the comment with the given comment.

void *removeNote* (const std::string& key)

Removes the Note object with the given key from the list of Note objects associated with the Face object.

void *removeInnerLoop* (EdgeLoopPtr& edgeLoopToRemove)

Removes the given EdgeLoop object's association with the Face object's holes (i.e. inner loops).

FacePtr

FacePtr is a type definition for a reference pointer to a Face object.

FacePtrList

FacePtrList is a type definition for a list of reference pointers to Face objects.

FacePtrMap

FacePtrMap is a type definition for an associative map between unique identifiers of Face objects and their reference pointers.

OrientedClosedShell Classes

OrientedClosedShell

The OrientedClosedShell class is a set of Face objects that form a closed shell that is oriented.

Public Attributes:

Real64* *centroid* ()

Centroid of the OrientedClosedShell object

Name *id* ()

Name object that identifies the OrientedClosedShell object.

std::string *name* ()

Name of the OrientedClosedShell object.

UInt32 *numberOfFaces* ()

Number of Face objects that compose the closed boundary of the OrientedClosedShell object.

UInt32 *numberOfNotes* ()

Number of Note objects associated with the OrientedClosedShell object.

UInt32 *numberOfSolidsUsingOrientedClosedShell* ()

Number of Solid objects that use the OrientedClosedShell object.

OrientedClosedShellPtr *opposite* ()

OrientedClosedShell object whose normal is opposite to the OrientedClosedShell object.

OrientationEnum *orientation* ()

Orientation (clockwise or counterclockwise) of OrientedClosedShell object.

Real64 *surfaceArea* ()

Surface Area of the OrientedClosedShell object

std::string& *uniqueId* ()

Unique identifier of the OrientedClosedShell object.

UInt32 *version* ()

Version number of the OrientedClosedShell object.

Real64 *volume* ()

Volume of the OrientedClosedShell object

Public Operations:

void *addNote* (const std::string& key, const std::string& comment)

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the OrientedClosedShell object.

void *debugPrint* ()

Prints the OrientedClosedShell object's member values to the error file cerr.

Logical *doesNoteExist* (const std::string& key)

Returns true if the Note object with the given key is associated with the OrientedClosedShell object, else false

FacePtr *getFace* (const std::string& name, Uint32 version = 0)

Returns a FacePtr to the Face object, which is specified by the given name and version, that is part of OrientedClosedShell object. If only the name is given, it is the unique id of the Face object.

const FacePtrList& *getFaces* ()

Returns a list of reference pointers (FacePtrList) to Face objects that compose the OrientedClosedShell object

const KeyList& *getKeysOfNotes* ()

Returns the keys of Note objects associated with the OrientedClosedShell object.

void *getNameAndVersion* (std::string& nameOut, Uint32& versionOut)

Returns the name and version number of the OrientedClosedShell object in the given arguments.

const Note& *getNote* (const std::string& key)

Returns the Note object with the given key.

const SolidPtrList& *getSolidsUsingOrientedClosedShell* ()

Returns a list of reference pointers (SolidPtrList) to Solid objects that use the OrientedClosedShell object

SolidPtr *getSolidUsingOrientedClosedShell* (const std::string& name, Uint32 version = 0)

Returns a SolidPtr to the Solid object, given the specified name and version, that the OrientedClosedShell object is a part of. If only the name is given, it is the unique id of the Solid object.

const UniqueldList& *getUidsOfFaces* ()

Returns a list (UniqueldList) of the unique identifiers to the Face objects that compose the OrientedClosedShell object.

const UniqueldList& *getUidsOfSolidsUsingOrientedClosedShell* ()

Returns a list (UniqueldList) of the unique identifiers to the Solid objects that use the OrientedClosedShell object.

void *modifyNote* (const std::string& key, const std::string& comment)

Modifies the Note object with the given key, that is associated with the OrientedClosedShell object, by replacing the comment with the given comment.

void *removeNote* (const std::string& key)

Removes the Note object with the given key from the list of Note objects associated with the OrientedClosedShell object.

OrientedClosedShellPtr

OrientedClosedShellPtr is a type definition for a reference pointer to an OrientedClosedShell object.

OrientedClosedShellPtrMap

OrientedClosedShellPtrMap is a type definition for an associative map between unique identifiers of OrientedClosedShell objects and their reference pointers.

OrientedClosedShellPtrList

OrientedClosedShellPtrList is a type definition for a list of reference pointers to OrientedClosedShell objects.

Pcurve Classes

Pcurve

A Pcurve object represents a parametric curve. A parametric curve is defined by means of a 2D curve in the parameter space of a surface. Its spline definition only contains geometry variables.

Derived from Spline

Public Attributes:

Logical *consideredStraight* (const Real64 tolerance = 0.001)

Returns true if the Pcurve object is straight within the given tolerance, else false.

Name *id* ()

Name object that identifies the Pcurve object.

Real64 *length3D* ()

Length of Pcurve object in 3D model space.

std::string *name* ()

Name of the Pcurve object.

UInt32 *numberOfMappedPpoints* ()

Number of Ppoint objects that are mapped to (lie on) the Pcurve object.

UInt32 *numberOfNotes* ()

Number of Note objects associated with the Pcurve object.

std::string& *uniqueId* ()

Unique identifier of the Pcurve object.

UInt32 *version* ()

Version number of the Pcurve object.

Public Operations:

void *addNote* (const std::string& key, const std::string& comment)

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the Pcurve object.

void *debugPrint* ()

Prints the Pcurve object's member values to the error file cerr.

Logical *doesNoteExist* (const std::string& key)

Returns true if the Note object with the given key is associated with the Pcurve object, else false

const std::vector<PcurveLocation>& *evlAtEqualArc* (UInt32 numLoc = 2)

Evaluate a Pcurve for surface (u,v) and (x, y, z) at equal arclengths given a number of points. A STL vector of PcurveLocation is returned.

const std::vector<PcurveLocation>& *evlAtEqualArc* (Real64 startValue, Real64 endValue, UInt32 numLoc = 2)

Evaluates a segment of the Pcurve for a given number of PcurveLocations which are at equal arclength spacing. A STL vector of PcurveLocations is returned.

const std::vector<PcurveLocation>& *evlAtEqualParametric* (UInt32 numLoc = 2)

Evaluate Pcurve for a given number of PcurveLocations which are at equal parametric spacing. A STL vector of PcurveLocations is returned.

const std::vector<PcurveLocation>& *evlAtEqualParametric* (Real64 startValue, Real64 endValue, UInt32 numLoc = 2)

Evaluates a segment of the Pcurve for a given number of PcurveLocations which are at equal parametric spacing. A STL vector of PcurveLocations is returned.

const CartesianLocation *evlForCartesianLoc* (Real64 s)

Evaluate Pcurve for (x, y, z) given (s). A CartesianLocation object is returned.

const PcurveLocation *evlForClosestPcrvLoc* (const Real64List& cartesianCoords, Real64 initialGuess, const Real64 tolerance, Real64& distance, Int32& numOfIterations, Int32& convergenceInfo)

For a given cartesian point (x, y, z), finds the closest location on the Pcurve object and return it as a PcurveLocation object.

const PcurveLocation *evlForPcurveLoc* (Real64 s)

Evaluate Pcurve for surface (u,v) and (x, y, z) given (s). A PcurveLocation object is returned.

SplineData *evlForSegmentCartesianSpline* (Real64 startValue, Real64 endValue, Logical reverseParWanted = false)

Evaluates the segment of a Pcurve given the start and end points of the segment for a cartesian spline. The domain (s) of the spline is parameterized from 0 to 1. The range of the spline is (x, y, z). The spline is returned as a SplineData object.

Real64 *evlForSegmentLength3D* (Real64 startValue, Real64 endValue)

Evaluates the 3D model space length of a segment of a Pcurve given the start and end points of the segment.

SplineData *evlForSegmentParametricSpline* (Real64 startValue, Real64 endValue, Logical reverseParWanted = false)

Evaluates the segment of a Pcurve given the start and end points of the segment for a parametric spline. The domain (s) of the spline is parameterized from 0 to 1. The range of the spline is (u, v). The spline is returned as a SplineData object.

const KeyList& *getKeysOfNotes* ()

Returns the keys of Note objects associated with the Pcurve object.

PpointPtr *getMappedPpoint* (const std::string& name, const UInt32 version = 0)

Returns a reference pointer (PpointPtr) to the Ppoint object that is mapped to the Pcurve object which was specified by name and version. If only the name is given, it is assumed to be the unique identifier of the Ppoint object.

const PpointPtrList& *getMappedPpoints* ()

Returns a list (PpointPtrList) of Ppoint objects that are mapped to the Pcurve object

void *getNameAndVersion* (std::string& nameOut, UInt32& versionOut)

Returns the name and version number of the Pcurve object in the given arguments.

const Note& *getNote* (const std::string& key)

Returns the Note object with the given key.

SurfacePtr *getSurface* ()

Returns a reference pointer (SurfacePtr) to the Surface object to which the Pcurve object is mapped.

const UniqueIdList& *getUidsOfMappedPpoints* ()

Returns a list (UniqueIdList) of unique identifiers of Ppoint objects that are mapped to the Pcurve object.

const std::string& *getUidOfSurface* ()

Returns a unique identifier to the Surface object that the Pcurve object maps to.

void *modifyNote* (const std::string& key, const std::string& comment)

Modifies the Note object with the given key, that is associated with the Pcurve object by replacing the comment with the given comment.

void *removeNote* (const std::string& key)

Removes the Note object with the given key from the list of Note objects associated with the Pcurve object.

PcurvePtr

PcurvePtr is a type definition for a reference pointer to a Pcurve object.

PcurvePtrList

PcurvePtrList is a type definition for a list of reference pointers to Pcurve objects.

PcurvePtrMap

PcurvePtrMap is a type definition for an associative map between unique identifiers of Pcurve objects and their reference pointers

Ppoint Classes

Ppoint

A Ppoint is a parametric point lying on a Pcurve.

Public Attributes:

Name *id* ()

Name object that identifies the Ppoint object.

Real64 *location* ()

Parametric value which specifies the location of the Ppoint object on the Pcurve object.

std::string *name* ()

Name of the Ppoint object.

UInt32 *numberOfEdgesIEnd* ()

Number of Edge objects the Ppoint object ends.

UInt32 *numberOfEdgesIStart* ()

Number of Edge objects the Ppoint object starts.

UInt32 *numberOfNotes* ()

Number of Note objects associated with the Ppoint object.

std::string& *uniqueId* ()

Unique identifier of the Ppoint object.

UInt32 *version* ()

Version number of the Ppoint object.

Public Operations:

void *addNote* (const std::string& key, const std::string& comment)

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the Ppoint object.

void *debugPrint* ()

Prints the Ppoint object's member values to the error file cerr.

Logical *doesNoteExist* (const std::string& key)

Returns true if the Note object with the given key is associated with the Ppoint object, else false

const CartesianLocation *evlForCartesianLoc* ()

Evaluate Ppoint for (x, y, z) at it's location. A CartesianLocation object is returned.

const PcurveLocation *evlForPcurveLoc* ()

Evaluate Ppoint for (s, u, v, x, y, z) at it's location. A PcurveLocation object is returned.

CoPointPtr *getCoPoint* ()

Returns a reference pointer (CoPointPtr) to the CoPoint object that uses the Ppoint object. If no CoPoint object is used by the Ppoint object, an invalid reference pointer is returned.

EdgePtr *getEdgeEnd* (const std::string& name, UInt32 version = 0)

Returns a reference pointer (EdgePtr) to the Edge object, given the specified name and version, that the Ppoint object ends. If only the name is given, it is the unique identifier of the Edge object.

const EdgePtrList& *getEdgesEnd* ()

Returns a list (EdgePtrList) of reference pointers to the Edge objects that the Ppoint object ends.

EdgePtr *getEdgeStart* (const std::string& name, UInt32 version = 0)

Returns a reference pointer (EdgePtr) to the Edge object, given the specified name and version, that the Ppoint object starts. If only the name is given, it is the unique identifier of the Edge object.

const EdgePtrList& *getEdgesStart* ()

Returns a list (EdgePtrList) of reference pointers to the Edge objects that the Ppoint object starts.

const KeyList& *getKeysOfNotes* ()

Returns the keys of Note objects associated with the Ppoint object.

void *getNameAndVersion* (std::string& nameOut, UInt32& versionOut)

Returns the name and version number of the Ppoint object in the given arguments.

const Note& *getNote* (const std::string& key)

Returns the Note object with the given key.

PcurvePtr *getPcurve* ()

Returns a reference pointer to the Pcurve object that the Pppoint object maps to.

const std::string& *getUidOfCoPoint* ()

Returns the unique identifier of the CoPoint object that uses the Ppoint object.
An empty string is returned if no CoPoint object uses the Ppoint object.

const UniqueIdList& *getUidsOfEdgesEnd* ()

Returns a list (UniqueIdList) of the unique identifiers to the Edge objects that the Ppoint object ends.

const UniqueIdList& *getUidsOfEdgesStart* ()

Returns a list (UniqueIdList) of the unique identifiers to the Edge objects that the Ppoint object starts.

const std::string& *getUidOfPcurve* ()

Returns the unique identifier to the Pcurve object that the Pppoint object maps to.

void *modifyNote* (const std::string& key, const std::string& comment)

Modifies the Note object with the given key, that is associated with the Ppoint object by replacing the comment with the given comment.

void *removeNote* (const std::string& key)

Removes the Note object with the given key from the list of Note objects associated with the Ppoint object.

PpointPtr

PpointPtr is a type definition for a reference pointer to a Ppoint object.

PpointPtrList

PpointPtrList is a type definition for a list of Ppoint objects.

PpointPtrMap

PpointPtrMap is a type definition for an associative map between unique identifiers of Ppoint objects and their reference pointers.

Solid Classes

Solid

The Solid class is a boundary represented solid.

Public Attributes:

Real64* *centroid* ()

Geometric centroid of the Solid object

Name *id* ()

Name object that identifies the Solid object.

std::string *name* ()

Name of the Solid object.

UInt32 *numberOfNotes* ()

Number of Note objects associated with the Solid object.

UInt32 *numberOfVoidShells* ()

Number of OrientedClosedShell objects that are voids in the Solid object.

Real64 *surfaceArea* ()

Outer surface area of the Solid object

std::string& *uniqueId* ()

Unique identifier of the Solid object.

UInt32 *version* ()

Version number of the Solid object.

Real64 *volume* ()

Volume of the Solid object

Public Operations:

void *addNote* (const std::string& key, const std::string& comment)

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the Solid object.

void *debugPrint* ()

Prints the Solid object's member values to the error file cerr.

Logical *doesNoteExist* (const std::string& key)

Returns true if the Note object with the given key is associated with the Solid object, else false

Logical *doesTopologicalViewExist* ()

Returns true if the Solid object is also a TopologicalView object, else false.

const KeyList& *getKeysOfNotes* ()

Returns the keys of Note objects associated with the Solid object.

void *getNameAndVersion* (std::string& nameOut, UInt32& versionOut)

Returns the name and version number of the Solid object in the given arguments.

const Note& *getNote* (const std::string& key)

Returns the Note object with the given key.

OrientedClosedShellPtr *getOuterShell* ()

Returns a reference pointer (OrientedClosedShellPtr) of the OrientedClosedShell object that bounds (i.e. outer shell) the Solid object.

TopologicalViewPtr *getTopologicalView* ()

Returns a reference pointer (TopologicalViewPtr) to the TopologicalView object that is represented by the Solid object. If no TopologicalView object is represented by the Solid object an invalid reference pointer is returned.

OrientedClosedShellPtr *getVoidShell* (const std::string& name, UInt32 version = 0)

Returns a reference pointer (OrientedClosedShellPtr) to the OrientedClosedShell object, given the specified name and version that represents a void in the Solid object. If only the name is given, it is the unique identifier of the OrientedClosedShell object.

const OrientedClosedShellPtrList& *getVoidShells* ()

Returns list (OrientedClosedShellPtrList) of reference pointers to OrientedClosedShell objects that represent voids in the Solid object.

const std::string& *getUidOfOuterShell* ()

Returns the unique identifier of the OrientedClosedShell object that bounds (i.e. outer shell) the Solid object.

const std::string& *getUidOfTopologicalView* ()

Returns the unique identifier of the TopologicalView object that the Solid object represents or an empty string if the Solid object does not represent a TopologicalView object.

const UniqueIdList& *getUidsOfVoidShells* ()

Returns list of unique identifiers of the OrientedClosedShell objects that represent voids in the Solid object.

void *modifyNote* (const std::string& key, const std::string& comment)

Modifies the Note object with the given key, that is associated with the Solid object, by replacing the comment with the given comment.

void *removeNote* (const std::string& key)

Removes the Note object with the given key from the list of Note objects associated with the Solid object.

void *removeVoidShell* (const OrientedClosedShellPtr& shellToRemove)

Removes the association between the Solid object and an OrientedClosedShell object that represents the void in the Solid object.

SolidPtr

SolidPtr is a type definition for a reference pointer to a Solid object.

SolidPtrList

SolidPtrList is a type definition for a list of reference pointers of Solid objects.

SolidPtrMap

SolidPtrMap is a type definition for an associative map between unique identifiers of Solid objects and their reference pointers.

Structure Classes

Structure

The Structure Class defines the highest level of physical representation for a design or part.

Public Attributes:

Name *id* ()

Name object that identifies the Structure object.

std::string *name* ()

Name of the Structure object.

UInt32 *numberOfCoEdges* ()

Number of CoEdge objects that are a part of the Structure object.

UInt32 *numberOfCommonViews* ()

Number of CommonView objects that are a part of the Structure object.

UInt32 *numberOfCoPoints* ()

Number of CoPoint objects that are a part of the Structure object.

UInt32 *numberOfEdges* ()

Number of Edge objects that are a part of the Structure object.

UInt32 *numberOfEdgeLoops* ()

Number of EdgeLoop objects that are a part of the Structure object.

UInt32 *numberOfFaces* ()

Number of Face objects that are a part of the Structure object.

UInt32 *numberOfMaterials* ()

Number of Material objects associated with the Structure object.

UInt32 *numberOfMaterialGroups* ()

Number of MaterialGroup objects associated with the Structure object.

UInt32 *numberOfNotes* ()

Number of Note objects associated with the Structure object.

UInt32 *numberOfOrientedClosedShells* ()

Number of OrientedClosedShell objects that are a part of the Structure object.

UInt32 *numberOfPcurves* ()

Number of Pcurve objects that are a part of the Structure object.

UInt32 *numberOfPpoints* ()

Number of Ppoint objects that are a part of the Structure object.

UInt32 *numberOfProperties* ()

Number of Property objects associated with the Structure object.

UInt32 *numberOfPropertyGroups* ()

Number of PropertyGroup objects associated with the Structure object.

UInt32 *numberOfSolids* ()

Number of Solid objects that are a part of the Structure object.

UInt32 *numberOfSurfaces* ()

Number of Surface objects that are a part of the Structure object.

UInt32 *numberOfTopologicalViews* ()

Number of TopologicalView objects that are a part of the Structure object.

std::string *uniqueId* ()

Unique identifier of the Structure object.

UInt32 *version* ()

Version number of the Structure object.

Public Operations:

void *addNote* (const std::string& key, const std::string& comment)

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the Structure object.

CoEdgePtr *createCoEdge* (const std::string& name, UInt32 version, const EdgePtrList& associatedEdges)

Creates a CoEdge object with the given name and version number and returns a reference pointer (CoEdgePtr) to the object. The CoEdge object is created from the list of given Edge objects.

CommonViewPtr *createCommonView* (const std::string& name, Uint32 version, const TopologicalViewPtrList& theTopologicalViews, const CommonViewPtrList& theCommonViews)

Creates a CommonView object with the given name, version number, TopologicalView objects, and CommonView objects. A reference pointer (CommonViewPtr) to this object is returned.

CoPointPtr *createCoPoint* (const std::string& name, Uint32 version, const PpointPtrList& memberPpoints, const CartesianLocation& loc)

Creates a CoPoint object with the given name, version number, and a list of Ppoint objects. The owner of the CoPoint object is set to the given Structure object. A reference pointer (CoPointPtr) to this object is returned.

EdgePtr *createEdge* (const std::string& name, Uint32 version, const PpointPtr& edgeStartPoint, const PpointPtr& edgeEndPoint)

Creates a Edge object with the given name and version number and returns a reference pointer (EdgePtr) to the object. The Edge object is created from the given Ppoint objects.

EdgeLoopPtr *createEdgeLoop* (const std::string& name, Uint32 version, const EdgePtrList& theEdges)

Creates an EdgeLoop object with the given name, version number, and a list of Edge objects. The owner of the EdgeLoop object is set to the given Structure object. A reference pointer (EdgeLoopPtr) to this object is returned.

FacePtr *createFace* (const std::string& name, Uint32 version, const EdgeLoopPtr& theOuterLoop, const EdgeLoopPtrList& theInnerLoops)

Creates an Face object with the given name, version number, the bounding loop and a list of EdgeLoop objects that represent holes in the face. The owner of the Face object is set to the given Structure object. A reference pointer (FacePtr) to this object is returned.

MaterialPtr *createMaterial* (const std::string& name, Uint32 version)

Creates a Material object with the given name and version. This object is associated with the Structure object.

MaterialGroupPtr *createMaterialGroup* (const std::string& name, Uint32 version, const MaterialPtrList& materialData, const MaterialGroupPtrList& materialGroupData)

Creates a MaterialGroup object with the given name, version, and a list of reference pointers to MaterialGroup and Material objects. This MaterialGroup object is associated with the Structure object.

OrientedClosedShellPtr *createOrientedClosedShell* (const std::string& name, Uint32 version, const FacePtrList& theFaces)

Creates an OrientedClosedShell object with the given name, version number, and a list of Face objects. The owner of the OrientedClosedShell object is set to the given Structure object. A reference pointer (OrientedClosedShellPtr) to this object is returned.

PcurvePtr *createPcurve* (const std::string& name, Uint32 version, const SurfacePtr& mapsToSurface, const SplineDomainVariableList& domainVars, const SplineRangeVariableList& rangeVars)

Creates a Pcurve object with the given name and version number and returns a reference pointer (PcurvePtr) to the object. The Pcurve object is created as a non-rational spline that represents a parametric curve that maps to the given surface.

PcurvePtr createPcurve (const std::string& name, Uint32 version, const SurfacePtr& mapsToSurface, const SplineDomainVariableList domainVars, const SplineRangeVariableList& rangeVars, const Real64List& weights)
Creates a Pcurve object with the given name and version number and returns a reference pointer (PcurvePtr) to the object. The Pcurve object is created as a rational spline that represents a parametric curve that maps to the given surface.

PcurvePtr createPcurve (const std::string& name, Uint32 version, const SurfacePtr& mapsToSurface, const Spline& spline)
Creates a Pcurve object with the given name and version number and returns a reference pointer (PcurvePtr) to the object. The Pcurve object is created from the given Spline object that represents a parametric curve that maps to the given surface.

const std::vector<std::pair<PcurvePtr, PcurvePtr> >& createPcurvesByIntersection (const std::string& pcrvARootName, const SurfacePtr& surfaceA, const std::string& pcrvBRootName, const SurfacePtr& surfaceB, Real64 toleranceIn = 0.0005)
Creates a STL vector of pairs of reference pointer to Pcurve objects created by intersecting two surfaces, A and B respectively. The first element of the pair is a Pcurve object that lies on Surface A, and the second element is a Pcurve object that lies on Surface B.

PpointPtr createPpoint (const std::string& name, Uint32 version, const PcurvePtr& mapsToPcurve, Real64 loc)
Creates a Ppoint object with the given name and version number and returns a reference pointer (PpointPtr) to the object. The Ppoint object is created from the given value that is on the given Pcurve object.

const std::vector<std::pair<PpointPtr, PpointPtr> >& createPpointsByIntersection (const std::string& ppntARootName, const PcurvePtr& pcurveA, const std::string& ppntBRootName, const PcurvePtr& pcurveB, Real64 toleranceIn = 0.0005)
Creates a STL vector of pairs of reference pointers to Ppoint objects created by intersecting two pcurves, A and B respectively. The first element of the pair is a reference pointer to a Ppoint object that lies on Pcurve A, and the second element is a reference pointer to a Ppoint object that lies on Pcurve B.

PropertyPtr createProperty (const std::string& name, Uint32 version, const PropertyDataPtr& propData)
Creates a Property object with the given name, version, and property data. This object is associated with the Structure object.

PropertyGroupPtr createPropertyGroup (const std::string& name, Uint32 version, const PropertyPtrList& propertyData, const PropertyGroupPtrList& propertyGroupData)
Creates a PropertyGroup object with the given name, version, and a list of reference pointers to PropertyGroup and Property objects. This PropertyGroup object is associated with the Structure object.

SolidPtr createSolid (const std::string& name, Uint32 version, const OrientedClosedShellPtr& theOuterShell, const OrientedClosedShellPtrList& theVoidShells)
Creates a Solid object from the given name, version number, bounding shell, and void shells. The owner of the Solid object is set to the given Structure object. A reference pointer (SolidPtr) to this object is returned.

SurfacePtr createSurface (const std::string& name, Uint32 version, const SplineDomainVariableList& domainVars, const SplineRangeVariableList& rangeVars)

Creates a Surface object with the given name and version number and returns a reference pointer (SurfacePtr) to the object. The Surface object is created as a non-rational spline that represents a surface.

SurfacePtr createSurface (const std::string& name, Uint32 version, const SplineDomainVariableList& domainVars, const SplineRangeVariableList& rangeVars, const Real64List& weights)

Creates a Surface object with the given name and version number and returns a reference pointer (SurfacePtr) to the object. The Surface object is created as a rational spline that represents a surface.

SurfacePtr createSurface (const std::string& name, Uint32 version, const Spline& spline)

Creates a Surface object with the given name and version number and returns a reference pointer (SurfacePtr) to the object. The Surface object is created with the given Spline object that represents a surface.

SurfacePtr createSurface (const std::string& name, Uint32 version, std::vector<Curve>& crvList, const Real64 toleranceln = 0.0001)

Creates a Surface object with the given name and version number and returns a reference pointer (SurfacePtr) to the object. The Surface object is created from a STL vector of Curve objects.

TopologicalViewPtr createTopologicalView (const std::string& name, Uint32 version, const SurfacePtr& surfView)

Creates a TopologicalView object with the given name, version number, and Surface object. A reference pointer (TopologicalViewPtr) to this object is returned.

TopologicalViewPtr createTopologicalView (const std::string& name, Uint32 version, const FacePtr& faceView)

Creates a TopologicalView object with the given name, version number, and Face object. A reference pointer (TopologicalViewPtr) to this object is returned.

TopologicalViewPtr createTopologicalView (const std::string& name, Uint32 version, const SolidPtr& solidView)

Creates a TopologicalView object with the given name, version number, and Solid object. A reference pointer (TopologicalViewPtr) to this object is returned.

void debugPrint ()

Prints the Structure object's member values to the error file cerr.

void destroyCommonView (const std::string& name, Uint32 version = 0)

Destroys the CommonView object with the given name and version number and all other objects dependent on it that is contained by this Structure object. If only the name is given, it is the unique identifier of the CommonView object.

void destroyCoEdge (const std::string& name, const Uint32 version = 0)

Destroys the CoEdge object with the given name and version number and all other objects dependent on it that is contained by this Structure object. If only the name is given, it is assumed to be the unique identifier of the CoEdge object.

void destroyCoPoint (const std::string& name, Uint32 version = 0)

Destroys the CoPoint object with the given name and version number and all other objects dependent on it that is contained by this Structure object. If only the name is given, it is assumed to be the unique identifier of the CoPoint object.

void *destroyEdge* (const std::string& name, Uint32 version = 0)

Destroys the Edge object with the given name and version number and all other objects dependent on it that is contained by this Structure object. If only the name is given, it is assumed to be the unique identifier of the Edge object.

void *destroyEdgeLoop* (const std::string& name, Uint32 version = 0)

Destroys the EdgeLoop object with the given name and version number and all other objects dependent on it that is contained by this Structure object. If only the name is given, it is assumed to be the unique identifier of the EdgeLoop object.

void *destroyFace* (const std::string& name, Uint32 version = 0)

Destroys the Face object with the given name and version number and all other objects dependent on it that is contained by this Structure object. If only the name is given, it is the unique identifier of the Face object.

void *destroyMaterial* (const std::string& name, Uint32 version = 0)

Destroys the Material object with the given name and version that is contained by this Structure object. If only the name is given, it is the unique identifier of the Material object.

void *destroyMaterialGroup* (const std::string& name, Uint32 version = 0)

Destroys the MaterialGroup object with the given name and version that is contained by this Structure object. If only the name is given, it is assumed to be the unique identifier of the MaterialGroup object.

void *destroyOrientedClosedShell* (const std::string& name, Uint32 version = 0)

Destroys the OrientedClosedShell object with the given name and version number and all other objects dependent on it that is contained by this Structure object. If only the name is given, it is assumed to be the unique identifier of the OrientedClosedShell object.

void *destroyPcurve* (const std::string& name, Uint32 version = 0)

Destroys the Pcurve object with the given name and version number and all other objects dependent on it that is contained by this Structure object. If only the name is given, it is assumed to be the unique identifier of the Pcurve object.

void *destroyPpoint* (const std::string& name, Uint32 version = 0)

Destroys the Ppoint object with the given name and version number and all other objects dependent on it that is contained by this Structure object. If only the name is given, it is assumed to be the unique identifier of the Ppoint object.

void *destroyProperty* (const std::string& name, Uint32 version = 0)

Destroys the Property object with the given name and version that is contained by this Structure object. If only the name is given, it is the unique identifier of the Property object.

void *destroyPropertyGroup* (const std::string& name, Uint32 version = 0)

Destroys the PropertyGroup object with the given name and version that is contained by this Structure object. If only the name is given, it is the unique identifier of the PropertyGroup object.

void *destroySolid* (const std::string& name, Uint32 version = 0)

Destroys the Solid object with the given name and version number and all other objects dependent on it that is contained by this Structure object. If only the name is given, it is assumed to be the unique identifier of the Solid object.

void *destroySurface* (const std::string& name, Uint32 version = 0)

Destroys the Surface object with the given name and version number and all other objects dependent on it that is contained by this Structure object. If only the name is given, it is the unique identifier of the Surface object.

void *destroyTopologicalView* (const std::string& name, Uint32 version = 0)

Destroys the TopologicalView object with the given name and version number and all other objects dependent on it that is contained by this Structure object. If only the name is given, it is assumed to be the unique identifier of the TopologicalView object.

Logical *doesCoEdgeExist* (const std::string& name, Uint32 version = 0)

Returns true if the CoEdge object with the given name and version number is associated with the Structure object, else false. If only the name is given, it is assumed to be the unique identifier of the CoEdge object.

Logical *doesCommonViewExist* (const std::string& name, Uint32 version = 0)

Returns true if the CommonView object with the given name and version number is associated with the Structure object, else false. If only the name is given, it is assumed to be the unique identifier of the CommonView object.

Logical *doesCoPointExist* (const std::string& name, Uint32 version = 0)

Returns true if the CoPoint object with the given name and version number is associated with the Structure object, else false. If only the name is given, it is assumed to be the unique identifier of the CoPoint object.

Logical *doesEdgeExist* (const std::string& name, Uint32 version = 0)

Returns true if the Edge object with the given name and version number is associated with the Structure object, else false. If only the name is given, it is the unique identifier of the Edge object.

Logical *doesEdgeLoopExist* (const std::string& name, Uint32 version = 0)

Returns true if the EdgeLoop object with the given name and version number is associated with the Structure object, else false. If only the name is given, it is assumed to be the unique identifier of the EdgeLoop object.

Logical *doesFaceExist* (const std::string& name, Uint32 version = 0)

Returns true if the Face object with the given name and version number is associated with the Structure object, else false. If only the name is given, it is the unique identifier of the Face object.

Logical *doesMaterialExist* (const std::string& name, Uint32 version = 0)

Returns true if the Material object with the given name and version number is associated with the Structure object, else false. If only the name is given, it is the unique identifier of the Material object.

Logical *doesMaterialGroupExist* (const std::string& name, Uint32 version = 0)

Returns true if the MaterialGroup object with the given name and version number is associated with the Structure object, else false. If only the name is given, it is the unique identifier of the MaterialGroup object.

Logical *doesNoteExist* (const std::string& key)

Returns true if the Note object with the given key is associated with the Structure object, else false

Logical *doesOrientedClosedShellExist* (const std::string& name, UInt32 version = 0)

Returns true if the OrientedClosedShell object with the given name and version number is associated with the Structure object, else false. If only the name is given, it is the unique identifier of the OrientedClosedShell object.

Logical *doesPcurveExist* (const std::string& name, UInt32 version = 0)

Returns true if the Pcurve object with the given name and version number is associated with the Structure object, else false. If only the name is given, it is the unique identifier of the Pcurve object.

Logical *doesPpointExist* (const std::string& name, UInt32 version = 0)

Returns true if the Ppoint object with the given name and version number is associated with the Structure object, else false. If only the name is given, it is the unique identifier of the Ppoint object.

Logical *doesPropertyExist* (const std::string& name, UInt32 version = 0)

Returns true if the Property object with the given name and version number is associated with the Structure object, else false. If only the name is given, it is the unique identifier of the Property object.

Logical *doesPropertyGroupExist* (const std::string& name, UInt32 version = 0)

Returns true if the PropertyGroup object with the given name and version number is associated with the Structure object, else false. If only the name is given, it is the unique identifier of the PropertyGroup object.

Logical *doesSolidExist* (const std::string& name, UInt32 version = 0)

Returns true if the Solid object with the given name and version number is associated with the Structure object, else false. If only the name is given, it is assumed to be the unique identifier of the Solid object.

Logical *doesSurfaceExist* (const std::string& name, UInt32 version = 0)

Returns true if the Surface object with the given name and version number is associated with the Structure object, else false. If only the name is given, it is the unique identifier of the Surface object.

Logical *doesTopologicalViewExist* (const std::string& name, UInt32 version = 0)

Returns true if the TopologicalView object with the given name and version number is associated with the Structure object, else false. If only the name is given, it is assumed to be the unique identifier of the TopologicalView object.

CoEdgePtr *getCoEdge* (const std::string& name, UInt32 version = 0)

Returns a reference pointer to the CoEdge object with the given name and version number that is contained by this Structure object. If only the name is given, it is assumed to be the unique identifier of the CoEdge object.

CommonViewPtr *getCommonView* (const std::string& name, UInt32 version = 0)

Returns a reference pointer to the CommonView object with the given name and version number that is contained by this Structure object. If only the name is given, it is the unique identifier of the CommonView object.

CoPointPtr *getCoPoint* (const std::string& name, UInt32 version = 0)

Returns a reference pointer to the CoPoint object with the given name and version number that is contained by this Structure object. If only the name is given, it is assumed to be the unique identifier of the CoPoint object.

EdgePtr *getEdge* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the Edge object with the given name and version number that is contained by this Structure object. If only the name is given, it is assumed to be the unique identifier of the Edge object.

EdgeLoopPtr *getEdgeLoop* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the EdgeLoop object with the given name and version number that is contained by this Structure object. If only the name is given, it is assumed to be the unique identifier of the EdgeLoop object.

FacePtr *getFace* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the Face object with the given name and version number that is contained by this Structure object. If only the name is given, it is the unique identifier of the Face object.

const KeyList& *getKeysOfNotes* ()

Returns the keys of Note objects associated with the Structure object.

MaterialPtr *getMaterial* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the Material object with the given name and version number that is contained by this Structure object. If only the name is given, it is the unique identifier of the Material object.

MaterialGroupPtr *getMaterialGroup* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the MaterialGroup object with the given name and version number that is contained by this Structure object. If only the name is given, it is the unique identifier of the MaterialGroup object.

void *getNameAndVersion* (std::string& nameOut, Uint32& versionOut)

Returns the name and version number of the Structure object in the given arguments.

const NameVersionPairList& *getNameVersionPairsOfCoEdges* ()

Returns the NameVersionPair of CoEdge objects that are associated with the Structure object.

const NameVersionPairList& *getNameVersionPairsOfCommonViews* ()

Returns the NameVersionPair of CommonView objects that are associated with the Structure object.

const NameVersionPairList& *getNameVersionPairsOfEdgeLoops* ()

Returns the NameVersionPair of EdgeLoop objects that are associated with the Structure object.

const NameVersionPairList& *getNameVersionPairsOfCoPoints* ()

Returns the NameVersionPair of CoPoint objects that are associated with the Structure object.

const NameVersionPairList& *getNameVersionPairsOfEdges* ()

Returns the NameVersionPair of Edge objects that are associated with the Structure object.

const NameVersionPairList& *getNameVersionPairsOfFaces* ()

Returns the NameVersionPair of Face objects that are associated with the Structure object.

const NameVersionPairList& *getNameVersionPairsOfMaterials* ()

Returns the NameVersionPair of Material objects that are associated with the Structure object.

- const NameVersionPairList& getNameVersionPairsOfMaterialGroups ()**
Returns the NameVersionPairs of MaterialGroup objects that are associated with the Structure object.
- const NameVersionPairList& getNameVersionPairsOfOrientedClosedShells ()**
Returns the NameVersionPairs of OrientedClosedShell objects that are associated with the Structure object.
- const NameVersionPairList& getNameVersionPairsOfPcurves ()**
Returns the NameVersionPairs of Pcurve objects that are associated with the Structure object.
- const NameVersionPairList& getNameVersionPairsOfPpoints ()**
Returns the NameVersionPairs of Ppoint objects that are associated with the Structure object.
- const NameVersionPairList& getNameVersionPairsOfProperties ()**
Returns the NameVersionPairs of Property objects that are associated with the Structure object.
- const NameVersionPairList& getNameVersionPairsOfPropertyGroups ()**
Returns the NameVersionPairs of PropertyGroup objects that are associated with the Structure object.
- const NameVersionPairList& getNameVersionPairsOfSolids ()**
Returns the NameVersionPairs of Solid objects that are associated with the Structure object.
- const NameVersionPairList& getNameVersionPairsOfSurfaces ()**
Returns the NameVersionPairs of Surface objects that are associated with the Structure object.
- const NameVersionPairList& getNameVersionPairsOfTopologicalViews ()**
Returns the NameVersionPairs of TopologicalView objects that are associated with the Structure object.
- const Note& getNote (const std::string& key)**
Returns the Note object with the given key.
- OrientedClosedShellPtr getOrientedClosedShell (const std::string& name, UInt32 version = 0)**
Returns a reference pointer to the OrientedClosedShell object with the given name and version number that is contained by this Structure object. If only the name is given, it is assumed to be the unique identifier of the OrientedClosedShell object.
- PcurvePtr getPcurve (const std::string& name, UInt32 version = 0)**
Returns a reference pointer to the Pcurve object with the given name and version number that is contained by this Structure object. If only the name is given, it is the unique identifier of the Pcurve object.
- PpointPtr getPpoint (const std::string& name, UInt32 version = 0)**
Returns a reference pointer to the Ppoint object with the given name and version number that is contained by this Structure object. If only the name is given, it is assumed to be the unique identifier of the Ppoint object.
- PropertyPtr getProperty (const std::string& name, UInt32 version = 0)**
Returns a reference pointer to the Property object with the given name and version number that is contained by this Structure object. If only the name is given, it is the unique identifier of the Property object.

PropertyGroupPtr *getPropertyGroup* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the PropertyGroup object with the given name and version number that is contained by this Structure object. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

SolidPtr *getSolid* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the Solid object with the given name and version number that is contained by this Structure object. If only the name is given, it is assumed to be the unique identifier of the Solid object.

SurfacePtr *getSurface* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the Surface object with the given name and version number that is contained by this Structure object. If only the name is given, it is assumed to be the unique identifier of the Surface object.

TopologicalViewPtr *getTopologicalView* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the TopologicalView object with the given name and version number that is contained by this Structure object. If only the name is given, it is assumed to be the unique identifier of the TopologicalView object.

const UniqueIdList& *getUidsOfCoEdges* ()

Returns the unique identifies of CoEdge objects that are associated with the Structure object.

const UniqueIdList& *getUidsOfCoPoints* ()

Returns the unique identifies of CoPoint objects that are associated with the Structure object.

const UniqueIdList& *getUidsOfCommonViews* ()

Returns the unique identifies of CommonView objects that are associated with the Structure object.

const UniqueIdList& *getUidsOfEdges* ()

Returns the unique identifies of Edge objects that are associated with the Structure object.

const UniqueIdList& *getUidsOfEdgeLoops* ()

Returns the unique identifies of EdgeLoop objects that are associated with the Structure object.

const UniqueIdList& *getUidsOfFaces* ()

Returns the unique identifies of Face objects that are associated with the Structure object.

const UniqueIdList& *getUidsOfMaterials* ()

Returns the unique identifies of Material objects that are associated with the Structure object.

const UniqueIdList& *getUidsOfMaterialGroups* ()

Returns the unique identifies of MaterialGroup objects that are associated with the Structure object.

const UniqueIdList& *getUidsOfOrientedClosedShells* ()

Returns the unique identifies of OrientedClosedShell objects that are associated with the Structure object.

const UniqueIdList& *getUidsOfPcurves* ()

Returns the unique identifies of Pcurve objects that are associated with the Structure object.

const UniqueIdList& *getUidsOfPpoints* ()

Returns the unique identifies of Ppoint objects that are associated with the Structure object.

const UniqueIdList& *getUidsOfProperties* ()

Returns the unique identifies of Property objects that are associated with the Structure object.

const UniqueIdList& *getUidsOfPropertyGroups* ()

Returns the unique identifies of PropertyGroup objects that are associated with the Structure object.

const UniqueIdList& *getUidsOfSolids* ()

Returns the unique identifies of Solid objects that are associated with the Structure object.

const UniqueIdList& *getUidsOfSurfaces* ()

Returns the unique identifies of Surface objects that are associated with the Structure object.

const UniqueIdList& *getUidsOfTopologicalViews* ()

Returns the unique identifies of TopologicalView objects that are associated with the Structure object.

void *modifyNote* (const std::string& key, const std::string& comment)

Modifies the Note object with the given key, that is associated with the Structure object, by replacing the comment with the given comment.

void *removeNote* (const std::string& key)

Removes the Note object with the given key from the list of Note objects associated with the Structure object.

StructurePtr

StructurePtr is a type definition for a reference pointer to a Structure object.

StructurePtrList

StructurePtrList is a type definition for a list of reference pointers to Structure objects.

StructurePtrMap

StructurePtrMap is a type definition for an associative map between unique identifiers of Structure objects and their reference pointers.

Surface Classes

Surface

The Surface class represents two dimensional geometry in Cartesian space.

Derived from Spline

Public Attributes:

Real64 *area* (Real64 tolerance = 0.00005)

Computes the area of the surface given the relative tolerance.

Real64List cartesianBounds ()

The cartesian bounding box of the surface. The bounding box is defined by its minimum point (xmin, ymin, zmin) and its maximum point (xmax, ymax, zmax). These values are store in a STL vector of real numbers.

Logical *consideredFlat* (const Real64 tolerance = 0.001)

Returns true if the Surface object is flat within the specified tolerance, else false.

Name *id* ()

Name object that identifies the Surface object.

std::string *name* ()

Name of the Surface object.

UInt32 *numberOfMappedPcurves* ()

Number of Pcurve objects that are mapped to (lie on) the Surface object.

UInt32 *numberOfNotes* ()

Number of Note objects associated with the Surface object.

std::string& *uniqueId* ()

Unique identifier of the Surface object.

UInt32 *version* ()

Version number of the Surface object.

Public Operations:

void *addNote* (const std::string& key, const std::string& comment)

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the Surface object.

void *debugPrint* ()

Prints the Surface object's member values to the error file cerr.

Logical *doesNoteExist* (const std::string& key)

Returns true if the Note object with the given key is associated with the Surface object, else false

Logical *doesTopologicalViewExist* ()

Returns true if the Surface object is also a TopologicalView object, else false.

const std::vector<SurfaceLocation>& *evlAtEqualArc* (UInt32 numULoc = 2, UInt32 numVLoc = 2)

Evaluates Surface object for a given number of SurfaceLocations which are at equal arclength spacing. A STL vector of SurfaceLocations is returned.

const std::vector<SurfaceLocation>& *evlAtEqualParametric* (UInt32 numULoc = 2, UInt32 numVLoc = 2)

Evaluates Surface for a given number of SurfaceLocations which are at equal parametric spacing. A STL vector of SurfaceLocations is returned.

const CartesianLocation *evlForCartesianLoc* (const Real64 u, const Real64 v)

Evaluate Surface for (x, y, z) given (u, v). A SurfaceLocation object is returned.

const SurfaceLocation *evlForClosestSurfLoc* (const Real64List& cartesianCoords, Real64List& initialGuess, Real64 tolerance, Real64& distance, Int32& numOfIterations, Int32& convergenceInfo)

For a given cartesian location (x, y, z), finds the closest location on the surface and return it as a SurfaceLocation object.

const std::vector<std::pair<SurfaceLocation, SurfaceLocation> >&

evlForEqualCorners (SurfacePtr surface, Real64 tolerance = 0.0001)

evlForEqualCorners tests the natural boundary corners of two surfaces for coincident Cartesian locations in the corners of the surface domain. The test occurs for each surface at (u,v) where u=(min,max) and v=(min,max), and considers equivalence to occur if the distance between the corners is within a specified tolerance. The return vector contains the pair of locations as SurfaceLocation on Surface 1 followed by the SurfaceLocation on Surface2.

const SurfaceLocation evlForSurfaceLoc (const Real64 u, const Real64 v)

Evaluate surface for the cartesian point (x,y,z) given the parametric point (u,v). A SurfaceLocation object is returned.

const KeyList& getKeysOfNotes ()

Returns the keys of Note objects associated with the Surface object.

PcurvePtr getMappedPcurve (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the Pcurve object with the specified name and version that maps to the Surface object. If only the name is given, it is assumed to be the unique identifier of the Pcurve object.

const PcurvePtrList& getMappedPcurves ()

Returns a list (PcurvePtrList) of reference pointers to all the Pcurve objects that are mapped to the Surface object.

void getNameAndVersion (std::string& nameOut, Uint32& versionOut)

Returns the name and version number of the Surface object in the given arguments.

const Note& getNote (const std::string& key)

Returns the Note object with the given key.

TopologicalViewPtr getTopologicalView ()

Returns a reference pointer to the TopologicalView object that is represented by the Surface object. If no TopologicalView object is represented by the Surface object an invalid reference pointer is returned.

const std::string& getTopologicalViewUid ()

Returns the unique identifier of the TopologicalView object that the Surface object represents or an empty string if the Surface object does not represent a TopologicalView object.

const UniqueldList& getUidsOfMappedPcurves ()

A list (UniqueldList) of unique identifiers of the Pcurve objects that are mapped to the Surface object.

void modifyNote (const std::string& key, const std::string& comment)

Modifies the Note object with the given key, that is associated with the Surface object by replacing the comment with the given comment.

void removeNote (const std::string& key)

Removes the Note object with the given key from the list of Note objects associated with the Surface object.

SurfacePtr

SurfacePtr is a type definition for a reference pointer to a Surface object.

SurfacePtrList

SurfacePtrList is a type definition for a list of reference pointers to Surface objects.

SurfacePtrMap

SurfacePtrMap is a type definition for an associative map between unique identifiers of Surface objects and their reference pointers.

TopologicalView Classes

TopologicalView

Topological Views are geometric representations that follow traditional CAD topologies. These include Surfaces, Manifold BREP Solids, and trimmed surfaces or Faces. A Topological View can be composed of 1 and only 1 geometric topology object. In this implementation that includes Solid, Face, and Surface objects. Because a Topological View is a geometric entity all properties associated with geometry are defined here and not in the subtypes.

Public Attributes:

Name *id* ()

Name object that identifies the TopologicalView object.

std::string *name* ()

Name of the TopologicalView object.

UInt32 *numberOfCommonViewsUsingTopologicalView* ()

Number of CommonView objects that use the TopologicalView object.

UInt32 *numberOfMaterials* ()

Number of Material objects associated with the TopologicalView object.

UInt32 *numberOfMaterialGroups* ()

Number of MaterialGroup objects associated with the TopologicalView object.

UInt32 *numberOfNotes* ()

Number of Note objects associated with the TopologicalView object.

TopologicalViewTypeEnum *objectType* ()

Specifies the type of object (Surface, Face, or Solid) that composes the TopologicalView object.

UInt32 *numberOfProperties* ()

Number of Property objects associated with the TopologicalView object.

UInt32 *numberOfPropertyGroups* ()

Number of PropertyGroup objects associated with the TopologicalView object.

std::string *uniqueId* ()

Unique identifier of the TopologicalView object.

UInt32 *version* ()

Version number of the TopologicalView object.

Public Operations:

void *addNote* (const std::string& key, const std::string& comment)

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the TopologicalView object.

MaterialPtr *createMaterial* (const std::string& name, Uint32 version)

Creates a Material object with the given name and version. This object is associated with the TopologicalView object.

MaterialGroupPtr *createMaterialGroup* (const std::string& name, Uint32 version, const MaterialPtrList& materialData, const MaterialGroupPtrList& materialGroupData)

Creates a MaterialGroup object with the given name, version, and a list of reference pointers to MaterialGroup and Material objects. This MaterialGroup object is associated with the TopologicalView object.

PropertyPtr *createProperty* (const std::string& name, Uint32 version, const PropertyDataPtr& propData)

Creates a Property object with the given name, version, and property data. This object is associated with the TopologicalView object.

PropertyGroupPtr *createPropertyGroup* (const std::string& name, Uint32 version, const PropertyPtrList& propertyData, const PropertyGroupPtrList& propertyGroupData)

Creates a PropertyGroup object with the given name, version, and a list of reference pointers to PropertyGroup and Property objects. This PropertyGroup object is associated with the TopologicalView object.

void *debugPrint* ()

Prints the TopologicalView object's member values to the error file cerr.

void *destroyMaterial* (const std::string& name, Uint32 version = 0)

Destroys the Material object with the given name and version that is contained by this TopologicalView object. If only the name is given, it is assumed to be the unique identifier of the Material object.

void *destroyMaterialGroup* (const std::string& name, Uint32 version = 0)

Destroys the MaterialGroup object with the given name and version that is contained by this TopologicalView object. If only the name is given, it is assumed to be the unique identifier of the MaterialGroup object.

void *destroyProperty* (const std::string& name, Uint32 version = 0)

Destroys the Property object with the given name and version that is contained by this TopologicalView object. If only the name is given, it is assumed to be the unique identifier of the Property object.

void *destroyPropertyGroup* (const std::string& name, Uint32 version = 0)

Destroys the PropertyGroup object with the given name and version that is contained by this TopologicalView object. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

Logical *doesMaterialExist* (const std::string& name, Uint32 version = 0)

Returns true if the Material object with the given name and version number is associated with the TopologicalView object, else false. If only the name is given, it is assumed to be the unique identifier of the Material object.

Logical *doesMaterialGroupExist* (const std::string& name, Uint32 version = 0)

Returns true if the MaterialGroup object with the given name and version number is associated with the TopologicalView object, else false. If only the name is given, it is assumed to be the unique identifier of the MaterialGroup object.

Logical *doesNoteExist* (const std::string& key)

Returns true if the Note object with the given key is associated with the TopologicalView object, else false

Logical *doesPropertyExist* (const std::string& name, Uint32 version = 0)

Returns true if the Property object with the given name and version number is associated with the TopologicalView object, else false. If only the name is given, it is assumed to be the unique identifier of the Property object.

Logical *doesPropertyGroupExist* (const std::string& name, Uint32 version = 0)

Returns true if the PropertyGroup object with the given name and version number is associated with the TopologicalView object, else false. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

const CommonViewPtrList& *getCommonViewsUsingTopologicalView* ()

Returns a list (CommonViewPtrList) of reference pointers to CommonView objects that use the TopologicalView object.

CommonViewPtr *getCommonViewUsingTopologicalView* (const std::string& name, Uint32 version = 0)

Returns a reference pointer (CommonViewPtr) to the CommonView object, given the specified name and version that use the TopologicalView object. If only the name is given, it is the unique identifier of the CommonView object.

FacePtr *getFace* ()

Returns a reference pointer (FacePtr) to the Face object that represents the TopologicalView object.

const KeyList& *getKeysOfNotes* ()

Returns the keys of Note objects associated with the TopologicalView object.

MaterialPtr *getMaterial* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the Material object with the given name and version number that is contained by this TopologicalView object. If only the name is given, it is the unique identifier of the Material object.

MaterialGroupPtr *getMaterialGroup* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the MaterialGroup object with the given name and version number that is contained by this TopologicalView object. If only the name is given, it is the unique identifier of the MaterialGroup object.

void *getNameAndVersion* (std::string& nameOut, Uint32& versionOut)

Returns the name and version number of the TopologicalView object in the given arguments.

const NameVersionPairList& *getNameVersionPairsOfMaterials* ()

Returns the NameVersionPairs of Material objects that are associated with the TopologicalView object.

- const NameVersionPairList& *getNameVersionPairsOfMaterialGroups* ()**
Returns the NameVersionPairs of MaterialGroup objects that are associated with the TopologicalView object.
- const NameVersionPairList& *getNameVersionPairsOfProperties* ()**
Returns the NameVersionPairs of Property objects that are associated with the TopologicalView object.
- const NameVersionPairList& *getNameVersionPairsOfPropertyGroups* ()**
Returns the NameVersionPairs of PropertyGroup objects that are associated with the TopologicalView object.
- const Note& *getNote* (const std::string& key)**
Returns the Note object with the given key.
- PropertyPtr *getProperty* (const std::string& name, Uint32 version = 0)**
Returns a reference pointer to the Property object with the given name and version number that is contained by this TopologicalView object. If only the name is given, it is assumed to be the unique identifier of the Property object.
- PropertyGroupPtr *getPropertyGroup* (const std::string& name, Uint32 version = 0)**
Returns a reference pointer to the PropertyGroup object with the given name and version number that is contained by this TopologicalView object. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.
- SolidPtr *getSolid* ()**
Returns a reference pointer to the Solid object that represents the TopologicalView object.
- SurfacePtr *getSurface* ()**
Returns a reference pointer (SurfacePtr) to the Surface object that represents the TopologicalView object.
- const UniqueIdList& *getUidsOfCommonViewsUsingTopologicalView* ()**
A list (UniqueIdList) of unique identifiers to CommonView objects that use the TopologicalView object.
- const std::string& *getUidOfFace* ()**
Returns the unique identifier of the Face object that represents the TopologicalView object
- const UniqueIdList& *getUidsOfMaterials* ()**
Returns the unique identifies of Material objects that are associated with the TopologicalView object.
- const UniqueIdList& *getUidsOfMaterialGroups* ()**
Returns the unique identifies of MaterialGroup objects that are associated with the TopologicalView object.
- const UniqueIdList& *getUidsOfProperties* ()**
Returns the unique identifies of Property objects that are associated with the TopologicalView object.
- const UniqueIdList& *getUidsOfPropertyGroups* ()**
Returns the unique identifies of PropertyGroup objects that are associated with the TopologicalView object.

const std::string& *getUidOfSolid* ()

Returns the unique identifier of the Solid object that represents the TopologicalView object.

const std::string& *getUidOfSurface* ()

Returns the unique identifier of the Surface object that represents the TopologicalView object.

void *modifyNote* (const std::string& key, const std::string& comment)

Modifies the Note object with the given key, that is associated with the TopologicalView object, by replacing the comment with the given comment.

void *removeNote* (const std::string& key)

Removes the Note object with the given key from the list of Note objects associated with the TopologicalView object.

TopologicalViewPtr

TopologicalViewPtr is a type definition for a reference pointer to a TopologicalView object.

TopologicalViewPtrList

TopologicalViewPtrList is a type definition for a list of reference pointers to TopologicalView objects.

TopologicalViewPtrMap

TopologicalViewPtrMap is a type definition for an associative map between unique identifiers of TopologicalView objects and their reference pointers.

LEAPS UTILITY CLASSES

The Utility package contains utility classes used by LEAPS.

ConnectionItem Classes

ConnectionItem

A ConnectionItem object is the basic object that composes a Connection object. The ConnectionItem object is a Component, System, or Connection object.

Public Attributes:

ConnectionItemTypeEnum *connectionItemType* ()

Specifies the type of connection item. A connection item can be a Component, System, or Connection object.

UInt32 *numberOfConnectionsUsingItem* ()

Number of Connection objects that use this ConnectionItem.

Public Operations:

***ConnectionItem* (const ComponentPtr& comp)**

Constructs a ConnectionItem object with the given Component object.

ConnectionItem (const ConnectionPtr& connec)

Constructs a ConnectionItem object with the given Connection object.

ConnectionItem (const SystemPtr& sys)

Constructs a ConnectionItem object with the given System object.

const std::string& getUidOfItem ()

Returns the unique identifier of the object that is contained by the ConnectionItem object.

ComponentPtr GetComponent ()

Returns a reference pointer (ComponentPtr) to the Component object that is the ConnectionItem object. If the ConnectionItem is not a Component, an invalid ComponentPtr is returned.

ConnectionPtr getConnection ()

Returns a reference pointer (ConnectionPtr) to the Connection object that is the ConnectionItem object. If the ConnectionItem is not a Connection, an invalid ConnectionPtr is returned.

const ConnectionPtrList& getConnectionsUsingItem ()

Returns a ConnectionPtrList of all Connection objects that use this ConnectionItem object.

ConnectionPtr getConnectionUsingItem (const std::string& name, UInt32 version = 0)

Returns a ConnectionPtr to the Connection object, given the specified name and version, that is uses the ConnectionItem object. If only the name is given, it is assumed to be the unique id of the Connection object.

SystemPtr getSystem ()

Returns a reference pointer (SystemPtr) to the System object that is the ConnectionItem object. If the ConnectionItem is not a System, an invalid SystemPtr is returned.

const UniqueIdList& getUidsOfConnectionsUsingItem ()

A list of unique ids of the Connection objects that use this ConnectionItem object.

bool isComponent ()

Returns true if the ConnectionItem object is a Component object , else false.

bool isConnection ()

Returns true if the ConnectionItem object is a Connection object , else false.

bool isSystem ()

Returns true if the ConnectionItem object is a System object , else false.

ConnectionItemList

ConnectionItemList is a type definition for a list of ConnectionItem objects.. It is defined as follows:

std::vector<Lps::ConnectionItem>.

Curve Classes

Curve

The Curve class that represents one dimension geometry in Cartesian space.

Derived from Spline

Public Attributes:

Name ***id()***

Name object that identifies the Curve object.

std::string ***name()***

Name of the Curve object.

UInt32 ***numberOfNotes()***

Number of Note objects associated with the Curve object.

std::string& ***uniqueId()***

Unique identifier of the Curve object.

UInt32 ***version()***

Version number of the Curve object.

Public Operations:

void ***addNote(const std::string& key, const std::string& comment)***

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the Curve object.

Curve(const std::string& name, UInt32 version, const std::vector<CartesianLocation>& pointList, Real64 toleranceIn = 0.0001)

Constructs a Curve object given a name, version number, and list of CartesianLocation objects.

Logical ***doesNoteExist(const std::string& key)***

Returns true if the Note object with the given key is associated with the Curve object, else false

std::vector<CurveLocation>& ***evlAtEqualArc(UInt32 numLoc = 2)***

Evaluate the Curve object for a given number of CurveLocations which are at equal arclength spacing. A STL vector of CurveLocations is returned.

std::vector<CurveLocation>& ***evlAtEqualParametric(UInt32 numLoc = 2)***

Evaluates the Curve object for a given number of CurveLocations which are at equal parametric spacing. A STL vector of CurveLocations is returned.

const CartesianLocation ***evlForCartesianLoc(Real64 s)***

Evaluate the Curve object for (x, y, z) given (s). A CartesianLocation object is returned.

const CurveLocation ***evlForCurveLoc(Real64 s)***

Evaluate the Curve object for (x, y, z) given (s). A CurveLocation object is returned.

const KeyList& ***getKeysOfNotes()***

Returns the keys of Note objects associated with the Curve object.

void ***getNameAndVersion(std::string& nameOut, UInt32& versionOut)***

Returns the name and version number of the Curve object in the given arguments.

const Note& ***getNote(const std::string& key)***

Returns the Note object with the given key.

void *modifyNote* (const std::string& key, const std::string& comment)
Modifies the Note object with the given key, that is associated with the Curve object, by replacing the comment with the given comment.

void *removeNote* (const std::string& key)
Removes the Note object with the given key from the list of Note objects associated with the Curve object.

Error Class

Error

Error provides error information for exception handling for LEAPS.

Derived from `std::exception`

Public Attributes:

char* *errorMessage* ()
Error message that describes the error encountered.

char* *routineName* ()
Name of routine in which the exception occurred.

Public Operations:

***Error* (const char* routineIn, const char* messageIn = "")**
Constructs an Error object given the name of routine in which the exception occurred and a message that describes the error encountered.

char* *what* ()
Error message that describes the error encountered.

Location Classes

CartesianLocation

The CartesianLocation class defines a location in Cartesian space as the coordinates (x,y,z).

Public Attributes:

Real64 *x* ()
x coordinate of the cartesian location (x, y, z)

Real64 *y* ()
y coordinate of the cartesian location (x, y, z)

Real64 *z* ()
z coordinate of the cartesian location (x, y, z)

Public Operations:

CartesianLocation (const Real64 xIn = 0.0, const Real64 yIn = 0.0, const Real64 zIn = 0.0)

Constructs a CartesianLocation object given with the given x, y, and z values.

CartesianLocation (const Real64List& coords)

Constructs a CartesianLocation object given an STL vector of Real64 values where x is at index 0, y is at index 1, and z is at index 2.

Real64 evlDistanceFromCartesianLoc (const CartesianLocation& cartPnt)

Returns the shortest distance from the given CartesianLocation object.

Real64 evlDistanceFromCartesianLoc (const Real64List& coords)

Returns the shortest distance from the given STL vector of Real64 values where x is at index 0, y is at index 1, and z is at index 2.

Real64 evlDistanceFromCartesianLoc (Real64 xCoord, Real64 yCoord, Real64 zCoord)

Returns the shortest distance from the given x, y, and z.

Real64List getCartesianLocation ()

Gets the cartesian location (x, y, z) where x is stored in index 0, y is stored in index 1, and z is stored at index 2.

Real64List getCartesianLocation (Real64& xOut, Real64& yOut, Real64& zOut)

Gets the cartesian location (x, y, z) where x is stored in index 0, y is stored in index 1, and z is stored at index 2. Furthermore, the cartesian location values is returned in the specified arguments.

CoEdgeLocation

The CoEdgeLocation class provides a list of PcurveLocations (s, u,v,x, y, z), evaluated at a parametric value (s) of the CoEdge object.

Public Attributes:

Real64 s ()

s is the parametric value at which the PcurvePoints were evaluated.

std::vector<PcurveLocation> pcurveLocationList ()

STL vector of PcurvePoints evaluated at parametric value (s)

Public Operations:

CoEdgeLocation (CoEdgePtr& coEdge, const Real64 s)

CoEdgePoint Constructor

const CartesianLocation getCartesianLocation (Real64& xOut, Real64& yOut, Real64& zOut)

Gets the cartesian location (x, y, z). Furthermore, the cartesian location values is returned in the specified arguments.

std::pair<Real64, std::vector<PcurveLocation> > getCoEdgeLocation (Real64& sOut, std::vector<PcurveLocation>& pcrvPntListOut)

Evaluates the CoEdge object at the parametric value (s) and returns a std::pair where the first element is the parametric value (s) and the second element is a STL vector of PcurveLocations (s, u,v,x, y, z).

CurveLocation

A CurveLocation is defined by the evaluation of a Curve object at the parametric value (s) for the Curve object's Cartesian coordinates (x,y,z).

Public Attributes:

Real64 s ()

s the parametric coordinate of the curve location

Real64 x ()

x coordinate of the curve location

Real64 y ()

y coordinate of the curve location

Real64 z ()

z coordinate of the curve location

Public Operations:

CurveLocation (const Curve& curveIn, Real64 sIn = 0.0)

CurveLocation Constructor

Real64List getCartesianLocation (Real64& xOut, Real64& yOut, Real64& zOut)

Gets the cartesian point (x, y, z) where x is stored in index 0, y is stored in index 1, and z is stored at index 2. Furthermore, the cartesian point values is returned in the specified arguments.

Real64List getCurveLocation (Real64& sOut, Real64& xOut, Real64& yOut, Real64& zOut)

Returns a STL vector of Real64 values that define the CurveLocation. s is stored at index 0, x is stored at index 1, y is stored at index 2, and z is stored at index 3. Furthermore, the values of the curve location is returned in the given arguments.

PcurveLocation

A PcurveLocation is defined by the evaluation of a Pcurve object at the parametric value (s) for the Pcurve object's surface parametric values (u,v) and the surface Cartesian coordinates (x,y,z) cooresponding to (u,v).

Public Attributes:

Real64 s ()

s the parametric coordinate of the pcurve location

Real64 u ()

U parametric value used to evaluate the SurfacePoint.

Real64 v ()

V parametric value used to evaluate the SurfacePoint.

Real64 x ()

x coordinate of the SurfacePoint object

Real64 y ()

y coordinate of the SurfacePoint object

Real64 z ()

z coordinate of the SurfacePoint object

Public Operations:

PcurveLocation (PcurvePtr& pcurve, const Real64 sIn = 0.0)

PcurvePoint Constructor

CartesianLocation getCartesianLocation ()

Returns the CartesianLocation object for the PcurveLocation..

Real64List getCartesianLocation (Real64& xOut, Real64& yOut, Real64& zOut)

Gets the cartesian point (x, y, z) where x is stored in index 0, y is stored in index 1, and z is stored at index 2. Furthermore, the cartesian point values is returned in the specified arguments.

Real64List getParametricLocation (Real64& sOut, Real64& uOut, Real64& vOut)

Returns a STL vector of Real64 values that define the parametric values of the PcurvePoint and SurfacePoint. s is stored at index 0, u is stored at index 1, and v is stored at index 2. Furthermore, the parametric values of the pcurve point and surface point is returned in the given arguments.

Real64List getPcurveLocation (Real64& sOut, Real64& uOut, Real64& vOut, Real64& xOut, Real64& yOut, Real64& zOut)

Returns a STL vector of Real64 values that define the PcurveLocation. s is stored at index 0, u is stored at index 1, v is stored at index 2, x is stored at index 3, y is stored at index 4, and z is stored at index 5. Furthermore, the values of the pcurve point is returned in the given arguments.

SurfaceLocation

The SurfaceLocation class defines a location on a Surface object. The location is derived by the evaluation of the Surface object at the Surface parametric values (u,v) for the Surface object's Cartesian coordinates (x,y,z).

Public Attributes:

Real64 u ()

U parametric value used to evaluate the SurfaceLocation.

Real64 v ()

V parametric value used to evaluate the SurfaceLocation.

Real64 x ()

x coordinate of the SurfaceLocation object

Real64 y ()

y coordinate of the SurfaceLocation object

Real64 z ()

z coordinate of the SurfaceLocation object

Public Operations:

SurfaceLocation (SurfacePtr& surface, Real64 uIn = 0.0, Real64 vIn = 0.0)

SurfaceLocation Constructor

Real64List *getCartesianLocation* (Real64& xOut, Real64& yOut, Real64& zOut)

Gets the cartesian point (x, y, z) where x is stored in index 0, y is stored in index 1, and z is stored at index 2. Furthermore, the cartesian point values is returned in the specified arguments.

Real64List *getParametricLocation* (Real64& uOut, Real64& vOut)

Returns a STL vector of Real64 values that define the parametric values of the SurfaceLocation. u is stored at index 0, and v is stored at index 1. Furthermore, the parametric values of the surface point is returned in the given arguments.

Real64List *getSurfaceLocation* (Real64& uOut, Real64& vOut, Real64& xOut, Real64& yOut, Real64& zOut)

Returns a STL vector of Real64 values that define the SurfaceLocation. u is stored at index 0, v is stored at index 1, x is stored at index 2, y is stored at index 3, and z is stored at index 4. Furthermore, the values of the surface point is returned in the given arguments.

Material Classes

The Materials Package contains classes used by the MaterialGroup Class.

Material

The Material class contain properties that pertain specifically to attributes of materials. These attributes do not depend on the geometry.

Public Attributes:

Name *id* ()

Name object that identifies the Material object.

std::string *name* ()

Name of the Material object.

UInt32 *numberOfMaterialGroupsUsingMaterial* ()

Number of MaterialGroup objects that use this Material object.

UInt32 *numberOfNotes* ()

Number of Note objects associated with the Material object.

UInt32 *numberOfProperties* ()

Number of Property objects associated with the Material object.

UInt32 *numberOfPropertyGroups* ()

Number of PropertyGroup objects associated with this object.

std::string& *uniqueId* ()

Unique identifier of the Material object.

UInt32 *version* ()

Version number of the Material object.

Public Operations:

void *addNote* (const std::string& key, const std::string& comment)

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the Material object.

PropertyPtr *createProperty* (const std::string& name, Uint32 version, const PropertyDataPtr& propData)

Creates a Property object with the given name, version, and property data. This object is associated with the Material object.

PropertyGroupPtr *createPropertyGroup* (const std::string& name, Uint32 version, const PropertyPtrList& propertyData, const PropertyGroupPtrList& propertyGroupData)

Creates a PropertyGroup object with the given name, version, and a list of reference pointers to PropertyGroup and Property objects. This PropertyGroup object is associated with the Material object.

void *debugPrint* ()

Prints the Material object's member values to the error file cerr.

void *destroyProperty* (const std::string& name, Uint32 version = 0)

Destroys the Property object with the given name and version that is contained by this Material object. If only the name is given, it is assumed to be the unique identifier of the Property object.

Logical *doesNoteExist* (const std::string& key)

Returns true if the Note object with the given key is associated with the Material object, else false

Logical *doesPropertyExist* (const std::string& name, Uint32 version = 0)

Returns true if the Property object with the given name and version number is associated with the Material object, else false. If only the name is given, it is assumed to be the unique identifier of the Property object.

Logical *doesPropertyGroupExist* (const std::string& name, Uint32 version = 0)

Returns true if the PropertyGroup object with the given name and version number is associated with the Material object, else false. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

const KeyList& *getKeysOfNotes* ()

Returns the keys of Note objects associated with the Material object.

const MaterialGroupPtrList& *getMaterialGroupsUsingMaterial* ()

Returns a list (MaterialGroupPtrList) of reference pointers to MaterialGroup objects that use the Material object.

MaterialGroupPtr *getMaterialGroupUsingMaterial* (const std::string& name, Uint32 version = 0)

Returns a reference pointer (MaterialGroupPtr) to the MaterialGroup object with the given name and version that uses the Material object. If only the name is given, it is assumed to be the unique identifier of the MaterialGroup object.

MaterialGroupPtr *getMaterialGroupUsingMatlAsAggregate* ()

Returns a MaterialGroupPtr to the Material object, that uses this Material object as its aggregate material. If the Material object is not an aggregate material, an invalid MaterialGroupPtr is returned.

void *getNameAndVersion* (std::string& nameOut, Uint32& versionOut)

Returns the name and version number of the Material object in the given arguments.

const NameVersionPairList& *getNameVersionPairsOfProperties* ()

Returns the NameVersionPairs of Property objects that are associated with the Material object.

const NameVersionPairList& getNameVersionPairsOfPropertyGroups ()

Returns the NameVersionPairs of PropertyGroup objects that are associated with the Material object.

const Note& getNote (const std::string& key)

Returns the Note object with the given key.

PropertyPtr getProperty (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the Property object with the given name and version number that is contained by this Material object. If only the name is given, it is assumed to be the unique identifier of the Property object.

PropertyGroupPtr getPropertyGroup (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the PropertyGroup object with the given name and version number that is contained by this Material object. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

const UniquedList& getUidsOfMaterialGroupsUsingMaterial ()

Returns a list (UniquedList) of unique identifiers to the MaterialGroup objects that use this Material object.

const std::string& getUidOfMaterialGroupUsingMatlAsAggregate ()

If the Material object is used as an aggregate material of a MaterialGroup object, the unique id of the MaterialGroup object is returned. If the Material object is not used as an aggregate material, an empty string is returned.

const UniquedList& getUidsOfProperties ()

Returns the unique identifies of Property objects that are associated with the Material object.

const UniquedList& getUidsOfPropertyGroups ()

Returns the unique identifies of PropertyGroup objects that are associated with the Material object.

void modifyNote (const std::string& key, const std::string& comment)

Modifies the Note object with the given key, that is associated with the Material object by replacing the comment with the given comment.

void removeNote (const std::string& key)

Removes the Note object with the given key from the list of Note objects associated with the Material object.

void removePropertyGroup (const std::string& name, Uint32 version = 0)

Removes the association of the PropertyGroup object with the specified name and version with the Material object.

MaterialPtr

MaterialPtr is a type definition for a reference pointer to a Material object.

MaterialPtrList

MaterialPtrList is a type definition for a list of reference pointers to Material objects. It is defined as follows:
std::vector<MaterialPtr>.

MaterialPtrMap

MaterialPtrMap is a type definition for an associative map between unique identifiers of Material objects and their reference pointers.. It is defined as follows:

```
std::map<std::string, MaterialPtr>
```

MaterialGroup

The MaterialGroup class provides the ability to group Material objects and/or other MaterialGroup objects together to compose a logical view of a material or a composite material. The MaterialGroup class also provides an aggregate material view of the MaterialGroup object.

Public Attributes:

Name **id ()**

Name object that identifies the MaterialGroup object.

std::string **name ()**

Name of the MaterialGroup object.

UInt32 **numberOfMaterials ()**

Number of Material objects that compose the MaterialGroup object.

UInt32 **numberOfMaterialGroups ()**

Number of MaterialGroup objects that compose the MaterialGroup object.

UInt32 **numberOfMaterialGroupsUsingMaterialGroup ()**

Number of MaterialGroup objects that use the MaterialGroup object.

UInt32 **numberOfNotes ()**

Number of Note objects associated with the MaterialGroup object.

UInt32 **numberOfProperties ()**

Number of Property objects associated with the MaterialGroup object.

UInt32 **numberOfPropertyGroups ()**

Number of PropertyGroup objects associated with the MaterialGroup object.

std::string& **uniqueId ()**

Unique identifier of the MaterialGroup object.

UInt32 **version ()**

Version number of the MaterialGroup object.

Public Operations:

void **addAggregateMaterial (const MaterialPtr& aggregateMatl)**

Adds the given Material object as the aggregate material of this MaterialGroup object.

void **addMaterial (const MaterialPtr& materialToAdd)**

Adds a Material object to the MaterialGroup object.

void **addMaterialGroup (const MaterialGroupPtr& groupToAdd)**

Adds a MaterialGroup object to the MaterialGroup object.

void **addNote (const std::string& key, const std::string& comment)**

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the MaterialGroup object.

PropertyPtr *createProperty* (const std::string& name, Uint32 version, const PropertyDataPtr& propData)

Creates a Property object with the given name, version, and property data. This object is associated with the MaterialGroup object.

PropertyGroupPtr *createPropertyGroup* (const std::string& name, Uint32 version, const PropertyPtrList& propertyData, const PropertyGroupPtrList& propertyGroupData)

Creates a PropertyGroup object with the given name, version, and a list of reference pointers to PropertyGroup and Property objects. This PropertyGroup object is associated with the MaterialGroup object.

void *debugPrint* ()

Prints the MaterialGroup object's member values to the error file cerr.

void *destroyProperty* (const std::string& name, Uint32 version = 0)

Destroys the Property object with the given name and version that is contained by this MaterialGroup object. If only the name is given, it is assumed to be the unique identifier of the Property object.

Logical *doesAggregateMaterialExist* ()

Returns true if the MaterialGroup object has an aggregate material associated with the it, else false.

Logical *doesMaterialExist* (const std::string& name, Uint32 version = 0)

Returns true if Material object exists, else false

Logical *doesMaterialGroupExist* (const std::string& name, Uint32 version = 0)

Returns true if the MaterialGroup object with the given name and version number is associated with the MaterialGroup object, else false. If only the name is given, it is the unique identifier of the MaterialGroup object.

Logical *doesNoteExist* (const std::string& key)

Returns true if the Note object with the given key is associated with the MaterialGroup object, else false

Logical *doesPropertyExist* (const std::string& name, Uint32 version = 0)

Returns true if the Property object with the given name and version number is associated with the MaterialGroup object, else false. If only the name is given, it is the unique identifier of the Property object.

Logical *doesPropertyGroupExist* (const std::string& name, Uint32 version = 0)

Returns true if the PropertyGroup object with the given name and version number is associated with the MaterialGroup object, else false. If only the name is given, it is the unique identifier of the PropertyGroup object.

MaterialPtr *getAggregateMaterial* ()

Returns a MaterialPtr to the Material object, that is the aggregate material of the MaterialGroup object.

const KeyList& *getKeysOfNotes* ()

Returns the keys of Note objects associated with the MaterialGroup object.

MaterialPtr *getMaterial* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the Material object with the given name and version number that is contained by this MaterialGroup object.

const MaterialPtrList& *getMaterials* ()

Returns a list (MaterialPtrList) of reference pointers to Material objects that compose the MaterialGroup object.

MaterialGroupPtr *getMaterialGroup* (const std::string& name, UInt32 version = 0)

Returns a reference pointer to the PropertyGroup object with the given name and version number that is contained by this PropertyGroup object. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

const MaterialGroupPtrList& *getMaterialGroups* ()

Returns a list (MaterialGroupPtrList) of reference pointers to MaterialGroup objects that compose the MaterialGroup object.

const MaterialGroupPtrList& *getMaterialGroupsUsingMaterialGroup* ()

Returns a list (MaterialGroupPtrList) of reference pointers to MaterialGroup objects that use the MaterialGroup object.

MaterialGroupPtr *getMaterialGroupUsingMaterialGroup* (const std::string& name, UInt32 version = 0)

Returns a reference pointer (MaterialGroupPtr) to the MaterialGroup object with the given name and version that uses the MaterialGroup object. If only the name is given, it is assumed to be the unique identifier of the MaterialGroup object.

void *getNameAndVersion* (std::string& nameOut, UInt32& versionOut)

Returns the name and version number of the MaterialGroup object in the given arguments.

const NameVersionPairList& *getNameVersionPairsOfProperties* ()

Returns the NameVersionPairs of Property objects that are associated with the MaterialGroup object.

const NameVersionPairList& *getNameVersionPairsOfPropertyGroups* ()

Returns the NameVersionPairs of PropertyGroup objects that are associated with the MaterialGroup object.

const Note& *getNote* (const std::string& key)

Returns the Note object with the given key.

PropertyPtr *getProperty* (const std::string& name, UInt32 version = 0)

Returns a reference pointer to the Property object with the given name and version number that is contained by this MaterialGroup object. If only the name is given, it is assumed to be the unique identifier of the Property object.

PropertyGroupPtr *getPropertyGroup* (const std::string& name, UInt32 version = 0)

Returns a reference pointer to the PropertyGroup object with the given name and version number that is contained by this MaterialGroup object. If only the name is given, it is assumed to be the unique identifier of the PropertyGroup object.

const std::string& *getUidOfAggregateMaterial* ()

The unique id of the Material object that is associated with the MaterialGroup object as the aggregate material is returned. An empty string is returned if there is no aggregate material.

const UniqueIdList& *getUidsOfMaterials* ()

Returns a list (UniqueIdList) of unique identifiers of Material objects that compose the MaterialGroup object.

const UniqueIdList& *getUidsOfMaterialGroups* ()

Returns a list (UniqueIdList) of unique identifiers of MaterialGroup objects that compose the MaterialGroup object.

const UniqueIdList& *getUidsOfMaterialGroupsUsingMaterialGroup* ()

Returns a list (UniqueIdList) of unique identifiers to the MaterialGroup objects that use this MaterialGroup object.

const UniqueIdList& *getUidsOfProperties* ()

Returns the unique identifies of Property objects that are associated with the MaterialGroup object.

const UniqueIdList& *getUidsOfPropertyGroups* ()

Returns the unique identifies of PropertyGroup objects that are associated with the MaterialGroup object.

void *modifyNote* (const std::string& key, const std::string& comment)

Modifies the Note object with the given key, that is associated with the MaterialGroup object, by replacing the comment with the given comment.

void *removeAggregateMaterial* ()

Removes the Material object that is associated with the MaterialGroup object as its aggregate material.

void *removeMaterial* (const std::string& name, Uint32 version = 0)

Removes the Material object with the specified name and version from this MaterialGroup object.

void *removeMaterialGroup* (const std::string& name, Uint32 version = 0)

Removes the MaterialGroup object with the specified name and version from this MaterialGroup object.

void *removeNote* (const std::string& key)

Removes the Note object with the given key from the list of Note objects associated with the MaterialGroup object.

void *removePropertyGroup* (const std::string& name, Uint32 version = 0)

Removes the association of the PropertyGroup object with the specified name and version with the MaterialGroup object.

MaterialGroupPtr

MaterialGroupPtr is a type definition for a reference pointer to a MaterialGroup object.

MaterialGroupPtrList

MaterialGroupPtrList is a type definition for a list of reference pointers to MaterialGroup objects. It is defined as follows:
std::vector<MaterialGroupPtr>.

MaterialGroupPtrMap

MaterialGroupPtrMap is a type definition for an associative map between unique identifiers of MaterialGroup objects and their reference pointers. It is defined as follows:

std::map<std::string, MaterialGroupPtr>

Name Classes

The Name Package contains all classes that are used by the Name Class.

DateTime

Date and time information needed to time stamp objects.

Public Attributes:

short *day* ()

Day in the month (1-31)

short *month* ()

Month of the year (1-12)

double *timeInSeconds* ()

Number of seconds that have elapsed since midnight for the given day.

time_t *unixTime* ()

The number of seconds elapsed since midnight (00:00:00), January 1, 1970, coordinated universal time.

short *year* ()

Number of years from AD

Public Operations:

void *debugPrint* ()

Prints the DateTime object's member values to the error file cerr.

std::string& *getTimeTimeString* ()

Returns a reference to a string that contains the DateTime in the form "1997/01/31.23:15:54 ". The string should be copied if the user wants it to be persistent.

void *setDateTime* ()

Sets the DateTime object's state to the current date and time.

Name

Name and audit information needed to identify LEAPS objects

Public Attributes:

std::string *completedBy* ()

User who marks the Name object as completed.

DateTime *completedOn* ()

The date/time the Name object was marked as completed.

std::string *createdBy* ()

User who created the Name object.

DateTime *createdOn* ()

The date/time the Name object was created.

std::string *globalId* ()

Global identifier of the Name object.

std::string *id* ()

The name of LEAPS object. The name must be alphanumeric and must begin with an alpha character.

std::string *modifiedBy* ()

User who last modified the Name object.

DateTime *modifiedOn* ()

The date/time the Name object was last modified.

unsigned long *numberOfNotes* ()

Number of notes that are associated with the Name object.

std::string *uniqueId* ()

Unique identifier of the Name object.

UInt32 *version* ()

Version number of the id.

Public Operations:

void *addNote* (const std::string& key, const std::string& comment)

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the Name object.

void *debugPrint* ()

Prints the Name object's member values to the error file cerr.

Logical *doesNoteExist* (const std::string& key)

Returns true if the Note object with the given key exists, else false

const KeyList& *getKeysOfNotes* ()

Returns the keys of Note objects associated with the Name object.

const Note& *getNote* (const std::string& key)

Returns the Note object with the given key.

const std::set<Note>& *getNotes* ()

Returns the set of Note objects associated with the Name object.

void *modifyNote* (const std::string& key, const std::string& comment)

Modifies the Note object with the given key by replacing the comment with the given comment.

void *removeNote* (const std::string& key)

Removes the Note object with the given key from the list of Note objects associated with the Name object.

Note

A Note associates user-defined information with a key and maintains audit information.

Public Attributes:

std::string *comment* ()

General information associated with the key of the Note object.

std::string *createdBy* ()

The user who created the Note object.

DateTime *createdOn* ()

The date/time the Note object was created.

std::string *key* ()

The key to which general information is associated.

std::string *modifiedBy* ()

The user who last modified the Note object.

DateTime *modifiedOn* ()

The date/time the Note object was last modified.

Public Operations:

void *debugPrint* ()

Prints the Note object's member values to the error file cerr.

NameValuePair Classes

NameValuePair

NameValuePair associates a label and a real number.

Public Attributes:

std::string *label* ()

Label that is associated with a real number.

Real64 *value* ()

A real number that is associated with the label.

Public Operations:

void *getNameValue* (std::string& label, Real64 value)

Gets the label and value of the NameValuePair object.

void *setNameValue* (const std::string& label, const Real64 value)

Sets the label and value of the NameValuePair object.

NameValuePairList

NameValuePairList is a type definition for a list of NameValuePair objects

Property Classes

The Properties Package contains classes used by the PropertyGroup Class.

IntegerScalar

The purpose of IntegerScalar is to provide a LEAPS data type that can store an integer as data.

Derived from [PropertyData](#)

[Public Attributes:](#)

PropertyDataTypeEnum *dataType* ()

The data type of the PropertyData object.

[Public Operations:](#)

void *debugPrint* ()

Prints the IntegerScalar object's member values to the error file cerr.

IntegerScalar (const Real64 value = 0.0)

Constructs a IntegerScalar object with the given real 64 bit number.

IntegerScalar (const Real32 value)

Constructs a IntegerScalar object with the given integer number.

IntegerScalar (const Int32 value)

Constructs a IntegerScalar object with the given 32 bit integer.

IntegerScalar (const UInt32 value)

Constructs a IntegerScalar object with the given unsigned 32 bit integer..

IntegerSTLVector

The purpose of IntegerSTLVector is to provide a LEAPS data type that can store a list of integers as data. A STL vector is used as the container for the integers.

Derived from [PropertyData](#), [Int32List](#)

[Public Attributes:](#)

PropertyDataTypeEnum *dataType* ()

The data type of the PropertyData object.

[Public Operations:](#)

IntegerSTLVector (const Int32* integerArray, const UInt32 integerArrayLength)

Constructs a IntegerSTLVectpr object with the given the array of integers of integerArrayLength.

IntegerSTLVector (const std::vector<Int32>& integerVector)

Constructs a IntegerSTLVectpr object with the given the STL vector of integers

void *debugPrint* ()

Prints the IntegerSTLVector object's member values to the error file cerr.

Property

Property objects associate data with a name and are used as metadata for representing an attribute of an object.

Public Attributes:

PropertyDataTypeEnum *dataType* ()

The data type of the PropertyData object that the Property object points to.

Name *id* ()

The id attribute is a Name object that identifies the Property object.

std::string *name* ()

Name of the Property object.

UInt32 *numberOfNotes* ()

Number of Note objects associated with the Property object.

UInt32 *numberOfPropertyGroupsUsingProperty* ()

Number of PropertyGroup objects that uses the Property object.

std::string& *uniqueId* ()

Unique string identifier of the Property object.

UInt32 *version* ()

Version number of the Property object.

Public Operations:

void *addNote* (const std::string& key, const std::string& comment)

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the Property object.

ToolPtr *createTool* (const std::string& name, UInt32 version)

Creates a Tool object which is associated with the Property object. This Tool object should be the tool that was used to determine the Property object.

void *debugPrint* ()

Prints the Property object's member values to the error file cerr.

Logical *doesNoteExist* (const std::string& key)

Returns true if the Note object with the given key is associated with the Property object, else false

Logical *doesToolExist* ()

Returns true if a Tool object was associated with the Property object, else false.

void *getData* (RealScalar& data)

Returns the RealScalar property data in the given argument.

void *getData* (RealSTLVector& data)

Returns the RealSTLVector property data in the given argument.

void *getData* (IntegerScalar& data)

Returns the IntegerScalar property data in the given argument.

void *getData* (IntegerSTLVector& data)

Returns the IntegerSTLVector property data in the given argument.

void *getData* (String& data)

Returns the String property data in the given argument.

void *getData* (StringSTLVector& data)

Returns the StringSTLVector property data in the given argument.

void *getData* (SplineData& data)

Returns the SplineData property data in the given argument.

PropertyDataPtr *getDataPtr* ()

Returns a reference pointer to the property data.

PropertyDataTypeEnum *getDataType* ()

Returns the data type that is contained by the Property object.

const KeyList& *getKeysOfNotes* ()

Returns the keys of Note objects associated with the Property object.

void *getNameAndVersion* (std::string& nameOut, Uint32& versionOut)

Returns the name and version number of the Property object in the given arguments.

const Note& *getNote* (const std::string& key)

Returns the Note object with the given key.

const PropertyGroupPtrList& *getPropertyGroupsUsingProperty* ()

Returns a list (PropertyGroupPtrList) of reference pointers to PropertyGroup objects that use the Property object.

PropertyGroupPtr *getPropertyGroupUsingProperty* (const std::string& name, Uint32 version = 0)

Returns a reference pointer (PropertyGroupPtr) to the PropertyGroup object with the given name and version that uses the Property object. If only the name is given, it is the unique identifier of the PropertyGroup object.

ToolPtr *getTool* ()

Returns a reference pointer to the Tool object that was used to determine the Property object with the Property object.

const UniqueIdList& *getUidsOfPropertyGroupsUsingProperty* ()

Returns a list (UniqueIdList) of unique identifiers to the PropertyGroup objects that use this Property object.

const std::string& *getUidOfTool* ()

Returns the unique identifier of the Tool object that was associated with the Property object or an empty string if there is no Tool object associated with the Property object.

void *modifyNote* (const std::string& key, const std::string& comment)

Modifies the Note object with the given key, that is associated with the Property object, by replacing the comment with the given comment.

void *removeNote* (const std::string& key)

Removes the Note object with the given key from the list of Note objects associated with the Property object.

PropertyPtr

PropertyPtr is a type definition for a reference pointer to a Property object.

PropertyPtrList

PropertyPtrList is a type definition for a list of reference pointers to Property objects.

PropertyPtrMap

PropertyPtrMap is a type definition for an associative map between unique identifiers of Property objects and their reference pointers.

PropertyData

PropertyData is an abstract class from which LEAPS data classes are derived from.

Public Attributes:

PropertyDataTypeEnum **dataType ()**

The data type of the PropertyData object.

Public Operations:

void **debugPrint ()**

A virtual member function that prints the PropertyData object's member values to the error file cerr.

PropertyDataPtr

PropertyDataPtr is a type definition for a reference pointer to a PropertyData object.

PropertyGroup

A PropertyGroup object associates a name with a group of Property objects and/or PropertyGroup objects. The purpose of this class is to allow a particular view of properties.

Public Attributes:

Name **id ()**

Name object that identifies the PropertyGroup object.

std::string& **uniqueId ()**

Unique identifier of the PropertyGroup object.

std::string **name ()**

Name of the PropertyGroup object.

UInt32 **version ()**

Version number of the PropertyGroup object.

UInt32 **numberOfNotes ()**

Number of Note objects associated with the PropertyGroup object.

UInt32 **numberOfPropertyGroups ()**

Number of PropertyGroup objects that are a part of the PropertyGroup object.

UInt32 **numberOfProperties ()**

Number of Property objects that are a part of the PropertyGroup object.

UInt32 **numberOfPropertyGroupsUsingPropertyGroup ()**

Number of PropertyGroup objects that use the PropertyGroup object.

Public Operations:

void *addNote* (const std::string& key, const std::string& comment)

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the PropertyGroup object.

void *addProperty* (const PropertyPtr& propertyToAdd)

Adds a Property object to this the PropetryGroup object.

void *addPropertyGroup* (const PropertyGroupPtr& groupToAdd)

Adds a PropertyGroup object this PropetryGroup object.

void *debugPrint* ()

Prints the PropertyGroup object's member values to the error file cerr.

Logical *doesNoteExist* (const std::string& key)

Returns true if the Note object with the given key is associated with the PropertyGroup object, else false

Logical *doesPropertyExist* (const std::string& name, Uint32 version = 0)

Returns true if Property object exists, else false. If only the name is given, it is the unique identifier of the Property object.

Logical *doesPropertyGroupExist* (const std::string& name, Uint32 version = 0)

Returns true if the PropertyGroup object with the given name and version number is part of this PropertyGroup object, else false. If only the name is given, it is the unique identifier of the PropertyGroup object.

const KeyList& *getKeysOfNotes* ()

Returns the keys of Note objects associated with the PropertyGroup object.

void *getNameAndVersion* (std::string& nameOut, Uint32& versionOut)

Returns the name and version number of the PropertyGroup object in the given arguments.

const Note& *getNote* (const std::string& key)

Returns the Note object with the given key.

const PropertyPtrList& *getProperties* ()

Returns a list (PropertyPtrList) of reference pointers to Property objects that compose the PropertyGroup object.

PropertyPtr *getProperty* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the Property object with the given name and version number that is contained by this PropertyGroup object.

PropertyGroupPtr *getPropertyGroup* (const std::string& name, Uint32 version = 0)

Returns a reference pointer to the PropertyGroup object with the given name and version number that is contained by this PropertyGroup object. If only the name is given, it is the unique identifier of the PropertyGroup object.

const PropertyGroupPtrList& *getPropertyGroups* ()

Returns a list (PropertyGroupPtrList) of reference pointers to PropertyGroup objects that compose the PropertyGroup object.

const PropertyGroupPtrList& *getPropertyGroupsUsingPropertyGroup* ()

Returns a list (PropertyGroupPtrList) of reference pointers to PropertyGroup objects that use the PropertyGroup object.

PropertyGroupPtr *getPropertyGroupUsingPropertyGroup* (const std::string& name, Uint32 version = 0)

Returns a reference pointer (PropertyGroupPtr) to the PropertyGroup object with the given name and version that uses the PropertyGroup object. If only the name is given, it is the unique identifier of the PropertyGroup object.

const UniqueldList& *getUidsOfProperties* ()

Returns a list (UniqueldList) of unique identifiers of Property objects that compose the PropertyGroup object.

const UniqueldList& *getUidsOfPropertyGroups* ()

Returns a list (UniqueldList) of unique identifiers of PropertyGroup objects that compose the PropertyGroup object.

const UniqueldList& *getUidsOfPropertyGroupsUsingPropertyGroup* ()

Returns a list (UniqueldList) of unique identifiers to the PropertyGroup objects that use this PropertyGroup object.

void *modifyNote* (const std::string& key, const std::string& comment)

Modifies the Note object with the given key, that is associated with the PropertyGroup object, by replacing the comment with the given comment.

void *removeNote* (const std::string& key)

Removes the Note object with the given key from the list of Note objects associated with the PropertyGroup object.

void *removeProperty* (const std::string& name, Uint32 version = 0)

Removes the Property object with the specified name and version from this PropertyGroup object.

void *removePropertyGroup* (const std::string& name, Uint32 version = 0)

Removes the PropertyGroup object with the specified name and version from this PropertyGroup object.

PropertyGroupPtr

PropertyGroupPtr is a type definition for a reference pointer to a PropertyGroup object.

PropertyGroupPtrList

PropertyGroupPtrList is a type definition for a list of reference pointers to PropertyGroup objects.

PropertyGroupPtrMap

PropertyGroupPtrMap is a type definition for an associative map between unique identifiers of PropertyGroup objects and their reference pointers.

RealScalar

The purpose of RealScalar is to provide a LEAPS data type that can store a real number as data provide operations of a real number.

Derived from [PropertyData](#)

[Public Attributes:](#)

PropertyDataTypeEnum *dataType* ()

The data type of the PropertyData object.

[Public Operations:](#)

void *debugPrint* ()

Prints the RealScalar object's member values to the error file cerr.

***RealScalar* (const Real64 value = 0.0)**

Constructs a RealScalar object with the given real number.

***RealScalar* (const Real32 value)**

Constructs a RealScalar object with the given real number.

***RealScalar* (const Int32 value)**

Constructs a RealScalar object with the given a 32 bit integer.

***RealScalar* (const UInt32 value)**

Constructs a RealScalar object with the given unsigned 32 bit integer..

RealSTLVector

The purpose of RealSTLVector is to provide a LEAPS data type that can store a list of real numbers as data. A STL vector is used as the container for the real numbers.

Derived from [PropertyData](#), [Real64List](#)

[Public Attributes:](#)

PropertyDataTypeEnum *dataType* ()

The data type of the PropertyData object.

[Public Operations:](#)

void *debugPrint* ()

Prints the RealSTLVector object's member values to the error file cerr.

***RealSTLVector* (const Real64* realArray, const UInt32 realArrayLength)**

Constructs a RealSTLVectpr object with the given the array of real numbers of realArrayLength.

***RealSTLVector* (const std::vector<Real64>& realVector)**

Constructs a RealSTLVectpr object with the given the STL vector of real numbers.

SplineData

Derived from PropertyData, Spline

Public Attributes:

PropertyDataTypeEnum *dataType* ()
The data type of the PropertyData object.

Public Operations:

void *debugPrint* ()
A virtual member function that prints the PropertyData object's member values to the error file cerr.

SplineData (const SplineDomainVariableList& domainVars, const SplineRangeVariableList& rangeVars, const Real64List& weights)
Construct a SplineData object that is a non-rational spline.

SplineData (const SplineDomainVariableList& domainVars, const SplineRangeVariableList& rangeVars)
Construct a SplineData object that is a rational spline.

SplineData (const Real64List& cArray, const CharStringList& domainLabels, const CharStringList& rangeLabels)
Construct a SplineData object with a CArray.

SplineData (Int32 dtnurbsHandle)
Construct a SplineData object given a DTNURBS handle.

SplineData (const std::vector<CartesianLocation>& pointList, Real64 toleranceln = 0.0001, Logical reverseParWanted = false)
Construct a SplineData object given a list of CartesianLocation objects.

String

The purpose of String is to provide a LEAPS data type that can store a string of characters as data.

Derived from PropertyData, CharString

Public Attributes:

PropertyDataTypeEnum *dataType* ()
The data type of the PropertyData object.

Public Operations:

void *debugPrint* ()
Prints the String object's member values to the error file cerr.

String (const std::string& charString = "")
Constructs a String object with the given string.

StringSTLVector

The purpose of StringSTLVector is to provide a LEAPS data type that can store a list of strings as data. A STL vector is used as the container for the strings.

Derived from PropertyData, CharStringList

Public Attributes:

PropertyDataTypeEnum *dataType* ()

The data type of the PropertyData object.

Public Operations:

void *debugPrint* ()

Prints the StringSTLVector object's member values to the error file cerr.

StringSTLVector (const Char8 charStringArray, const UInt32 charStringArrayLength)**

Constructs a StringSTLVector object with the given array of strings

StringSTLVector (const std::vector<std::string>& stringVector)

Constructs a StringSTLVector object with the given STL vector of strings.

Spline Classes

Spline

The spline class is a supertype of those classes that represent geometry, topology, and behavior models. It is non-dimensional in domain and range.

Public Attributes:

Int32 *dtNurbsHandle* ()

The Spline class uses the DtNurbs Spline library. This attribute is the handle that identifies the DtNurbs entity used by the DtNurbs Spline library.

UInt32 *numberOfRangeVars* ()

Number of range (dependent) variables associated with the spline.

UInt32 *numberOfDomainVars* ()

Number of domain (independent) variables associated with the spline.

UInt32 *numberOfWeights* ()

Number of weights that are associated with the spline.

Logical *isRational* ()

Indicates whether the spline is rational or nonrational.

SplineDomainVariableList *domainVariables* ()

Domain (independent) variables of the spline.

SplineRangeVariableList *rangeVariables* ()

Range (dependent) variables of the spline.

Real64List *weightData* ()

Weights of the rational spline.

Public Operations:

void create (const SplineDomainVariableList& domainVars, const SplineRangeVariableList& rangeVars, const Real64List& weights)

Initializes a Spline object which is a non-uniform rational spline. If the given weights size is 0, the spline is assumed to be non-rational.

const std::vector<Real64List>& evlAtEqualParametric (const Uint32List& numLocs)

Evaluate Spline at a number of points at equal parametric spacing given the number of points. . A STL vector of Real64List is returned.

const Real64List& evlSpline (const Real64List& domainValues)

Evaluate spline range values given spline domain values. The range values are returned in a reference to a Real64List. The spline domain variables are given in a Real64List.

const Real64List& getArray ()

Returns a reference to a DTNURBS spline communication array that represents the Spline object. The life of the reference is only valid until the next call to this method.

const SplineDomainVariable& getDomainVar (Uint32 position = 1)

Return a reference to the SplineDomainVariable object given the specified position.

const SplineRangeVariable& getRangeVar (Uint32 position = 1)

Return a reference to the SplineRangeVariable object given the specified position.

Real64 getWeight (Uint32 position = 1)

Return the weight at the specified position.

SplineDomainVariable

The SplineDomainVariable class defines the function domain variables used by the Spline class. In spline terminology the domain variables contain the parametric information.

Public Attributes:

Real64 highValue ()

Value of maximum knot

Real64List knots ()

Knots of the spline that are associated with the independent (domain) variable.

std::string label ()

Label that describes the independent (domain) variable

Real64 lowValue ()

Value of minimum knot.

Uint32 numberOfKnots ()

Number of knots associated with the independent (domain) variable that are part of the spline.

Uint32 order ()

Order of the spline for the independent (domain) variable.

Public Operations:

void *debugPrint* ()

Prints the objects member values to the error file cerr.

void *create* (const Real64List& knots, Uint32 order, const std::string& label = "")

Creates a SplineDomainVariable object with the given knots, order, and label.

Real64 *getKnot* (const Uint32 position = 1)

Returns the knot specified by position.

SplineDomainVariableList

SplineDomainVariableList is a type definition for a list of SplineDomainVariable objects.. It is defined as follows:

std::vector<Lps::SplineDomainVariable>.

SplineRangeVariable

The SplineRangeVariable class defines the function range variables used by the Spline class. In spline terminology the range variables contain control points information.

Public Attributes:

Real64List *controlPoints* ()

Control points associated with the range (dependent) variable that control the spline.

std::string *label* ()

Label that describes the dependent (range) variable

Uint32 *numberOfControlPnts* ()

Number of control points. Control points control the spline.

Public Operations:

void *debugPrint* ()

Prints the objects member values to the error file cerr.

void *create* (const std::vector<Real64>& cntrlPnts, const std::string& label = "")

Creates a SplineRangeVariable object with the given control points and label.

Real64 *getControlPnt* (const Uint32 position = 1)

Returns the control point (coefficient) at the specified position

SplineRangeVariableList

SplineRangeVariableList is a type definition for a list of SplineRangeVariable objects.. It is defined as follows:

std::vector<Lps::SplineRangeVariable>.

Tool Classes

Tool

Public Attributes:

Name *id* ()

Name object that identifies the Tool object.

std::string *name* ()

Name of the Tool object.

UInt32 *numberOfNotes* ()

Number of Note objects associated with the Tool object.

std::string& *uniqueId* ()

Unique identifier of the Tool object.

UInt32 *version* ()

Version number of the Tool object.

Public Operations:

void *addNote* (const std::string& key, const std::string& comment)

Creates a Note object with the given key and comment. The Note object is added to the list of Note objects associated with the Tool object.

void *debugPrint* ()

Prints the Tool object's member values to the error file cerr.

Logical *doesNoteExist* (const std::string& key)

Returns true if the Note object with the given key is associated with the Tool object, else false.

const KeyList& *getKeysOfNotes* ()

Returns the keys of Note objects associated with the Tool object.

void *getNameAndVersion* (std::string& nameOut, UInt32& versionOut)

Returns the name and version number of the Tool object in the given arguments.

const Note& *getNote* (const std::string& key)

Returns the Note object with the given key.

void *modifyNote* (const std::string& key, const std::string& comment)

Modifies the Note object with the given key, that is associated with the Tool object by replacing the comment with the given comment.

void *removeNote* (const std::string& key)

Removes the Note object with the given key from the list of Note objects associated with the Tool object.

ToolPtr

ToolPtr is a type definition for a reference pointer to a Tool object.

ToolPtrList

ToolPtrList is a type definition for a list of reference pointers to Tool objects.

ToolPtrMap

ToolPtrMap is a type definition for an associative map between unique identifiers of Tool objects and their reference pointers.